EECE 5639 Computer Vision I

Lecture 7

Canny Edge detector. Laplacian. Modern Edge detection. Corners

Project 2 is out.

First Midterm: February 12

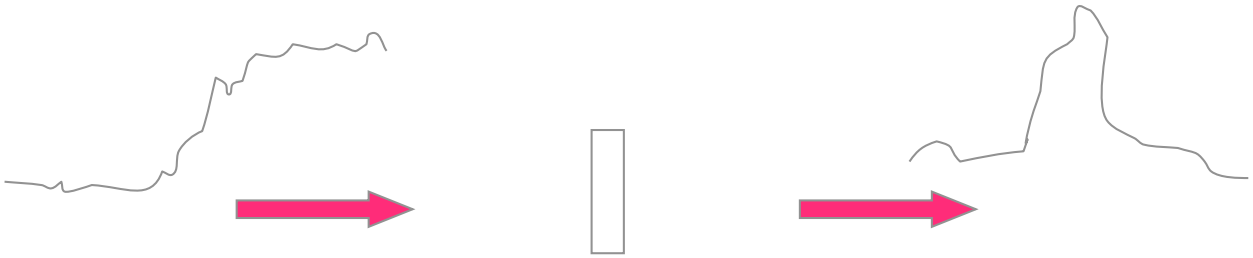
Next Class

Sift Features, Hough transform

Canny's Performance Criteria

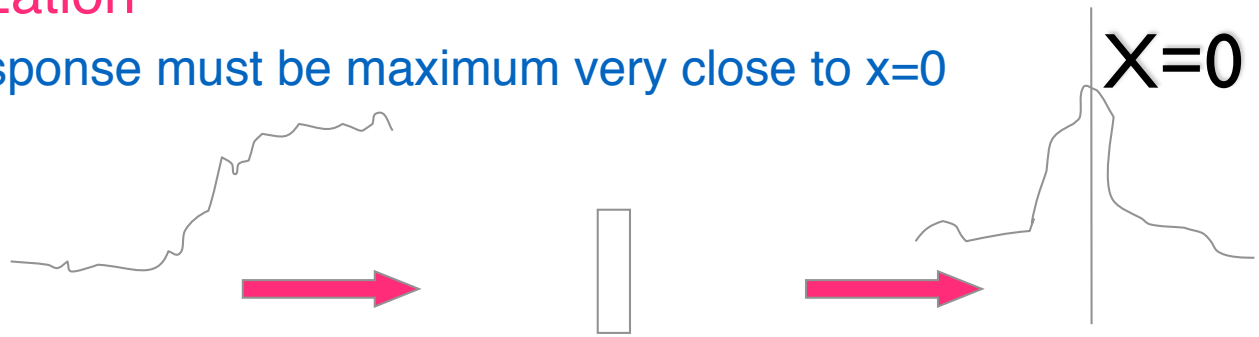
Good detection

The filter must have a stronger response at the edge location (x=0) than to noise



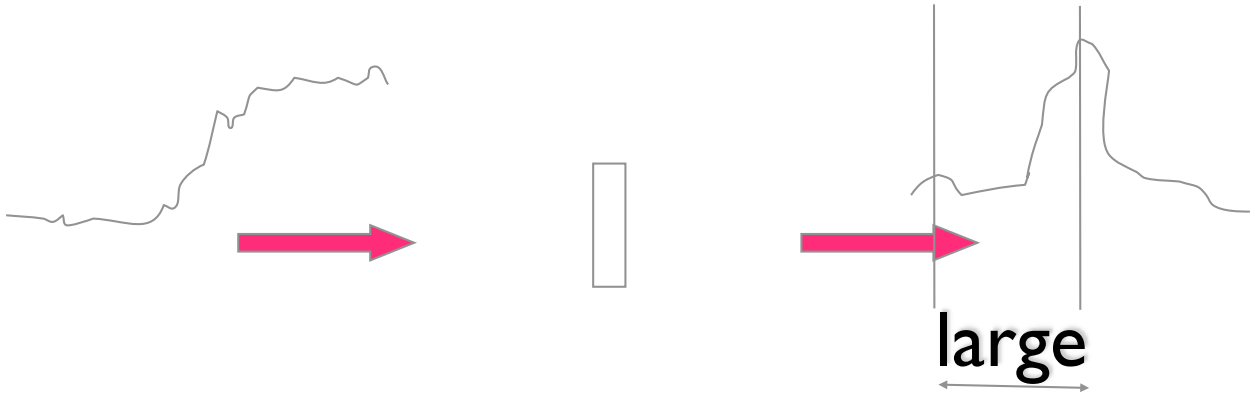
Good Localization

The filter response must be maximum very close to x=0



Low False Positives

There should be only one maximum in a reasonable neighborhood of x=0

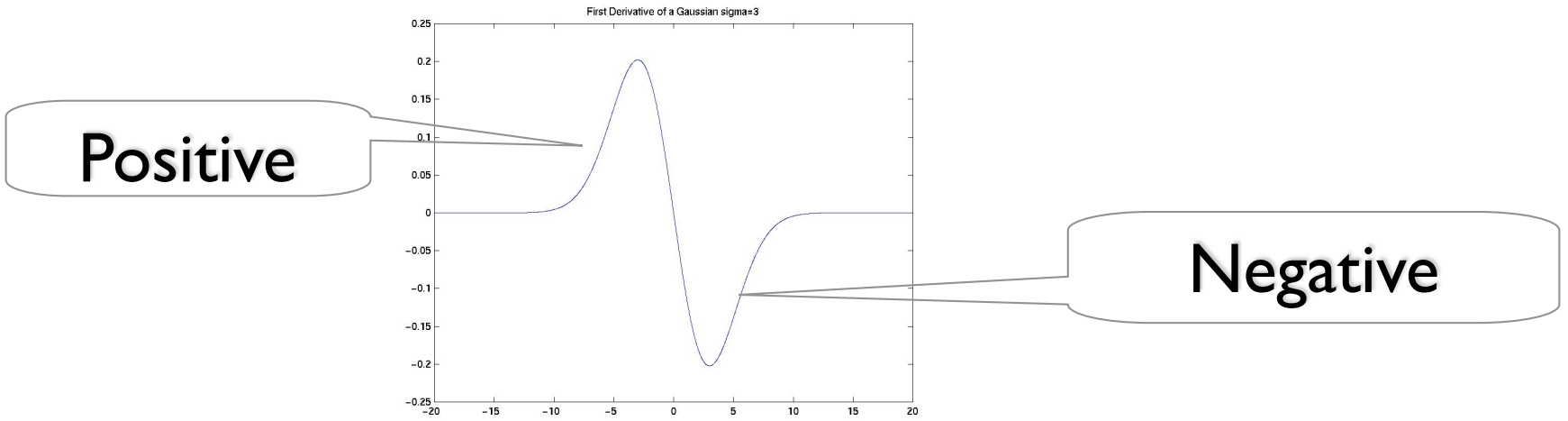


Canny Edge Detector

Canny found a linear, continuous filter that maximized the three given criteria.

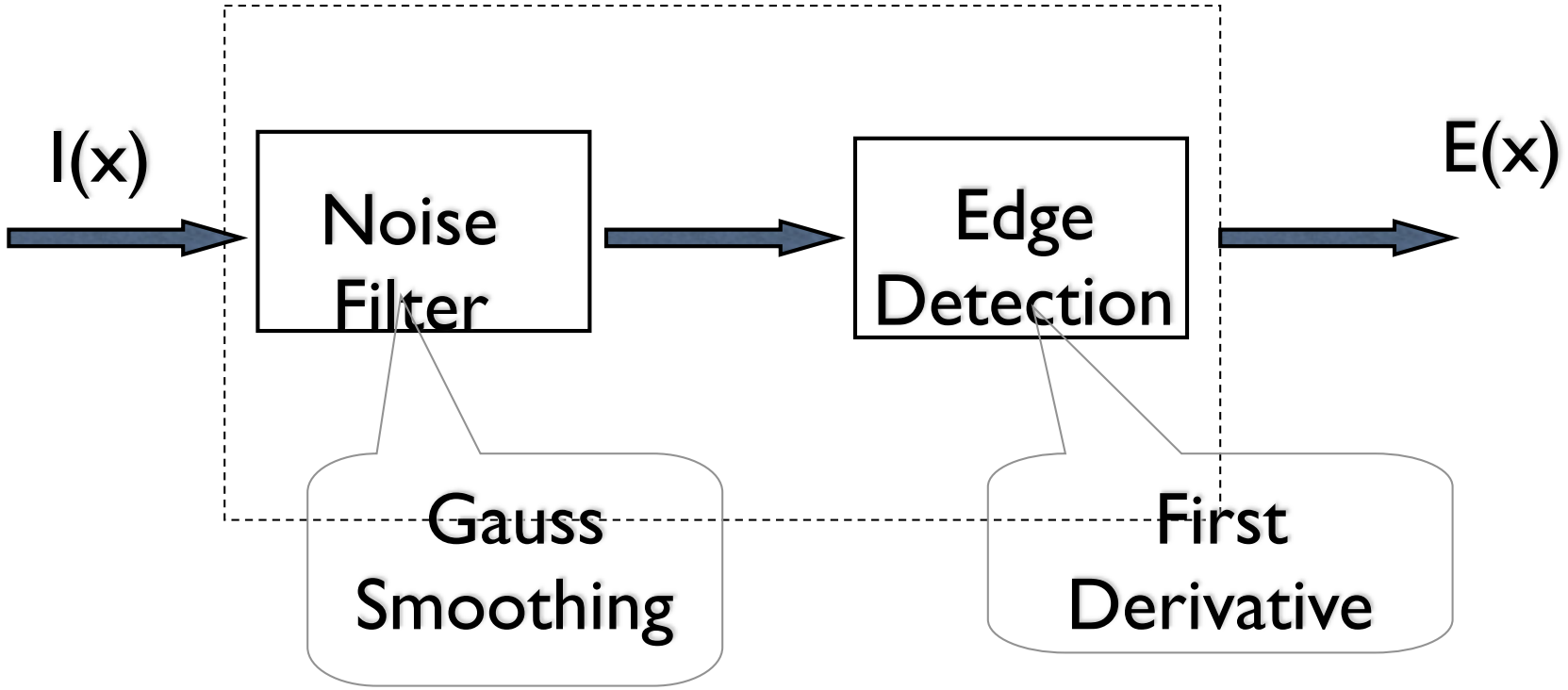
There is no close-form solution for the optimal filter.

However, it looks VERY SIMILAR to the derivative of a Gaussian.



$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

Another interpretation:



$$E(x) = \frac{d(I(x) * G(x))}{dx} = I(x) * \frac{dG(x)}{dx}$$

Algorithm CANNY_ENHANCER

The input is image I; G is a zero mean Gaussian filter (std = σ)

1. J = I * G (smoothing)
2. For each pixel (i,j): (edge enhancement)

   Compute the image gradient

   $\nabla J(i,j) = (J_x(i,j), J_y(i,j))'$

   1. Estimate edge strength

      $e_s(i,j) = (J_x^2(i,j) + J_y^2(i,j))^{1/2}$

   Estimate edge orientation

   $e_o(i,j) = \arctan(J_x(i,j)/J_y(i,j))$

The output are images E_s and E_o

CANNY_ENHANCER

The output image E_s has the magnitudes of the smoothed gradient.

Sigma determines the amount of smoothing.

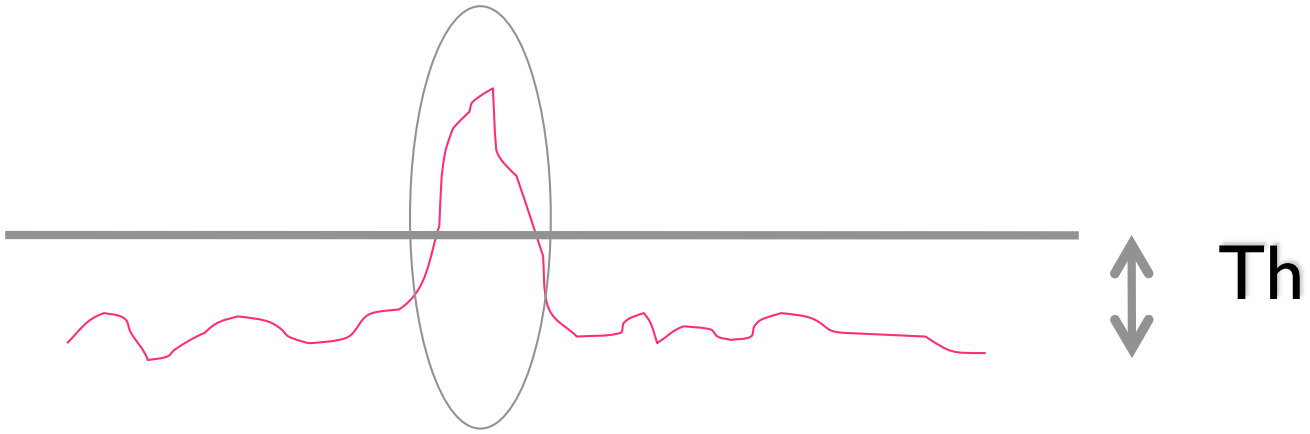
E_s has large values at edges

Edge ENHANCER

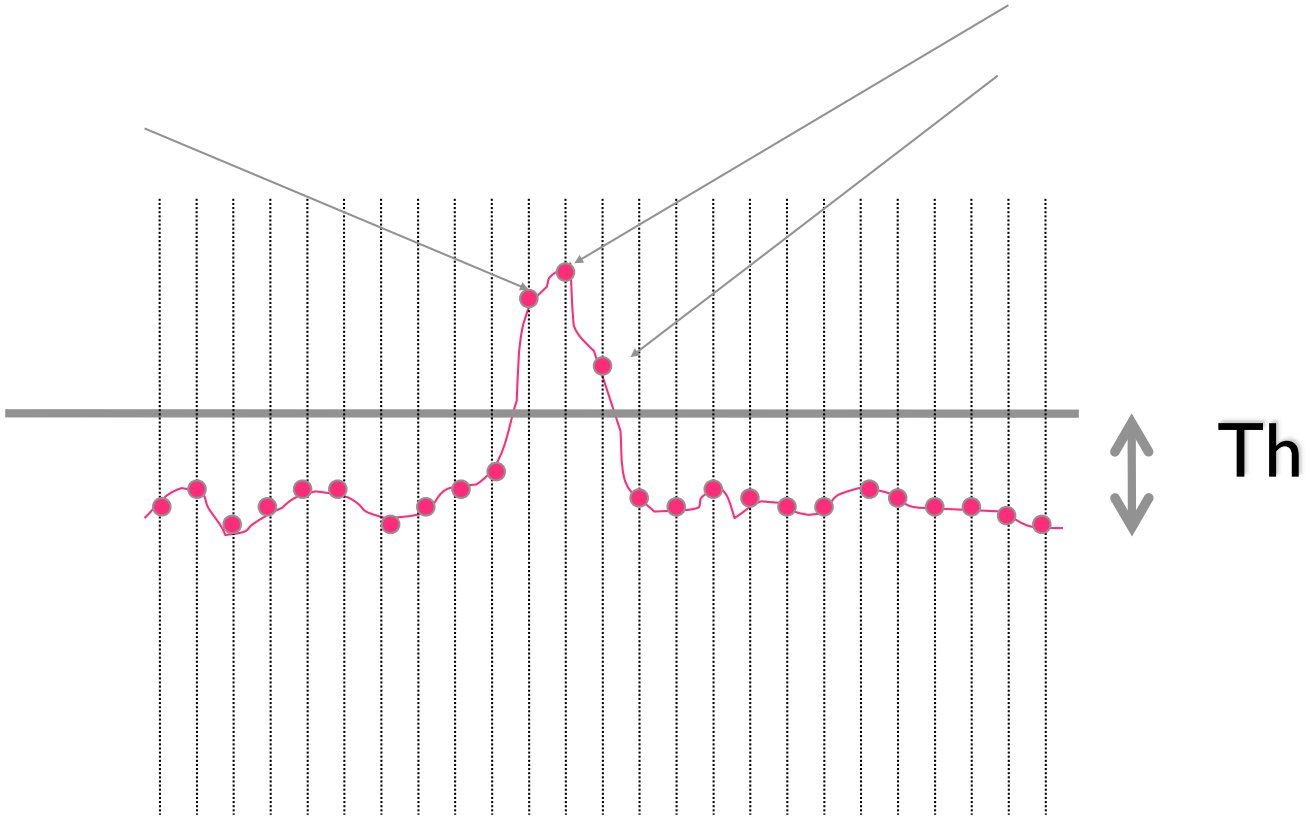
How do we “detect” edges?

E_s has large values at edges:

Find local maxima



… but it also may have wide ridges around the local maxima (large values around the edges)



NONMAX_SUPRESSION

The inputs are E_s & E_o (outputs of CANNY_ENHANCER)

Consider 4 directions D={ 0,45,90,135} wrt x

For each pixel (i,j) do:

1. Find the direction $d \in D$ s.t. $d \cong E_o(i,j)$ (normal to the edge)

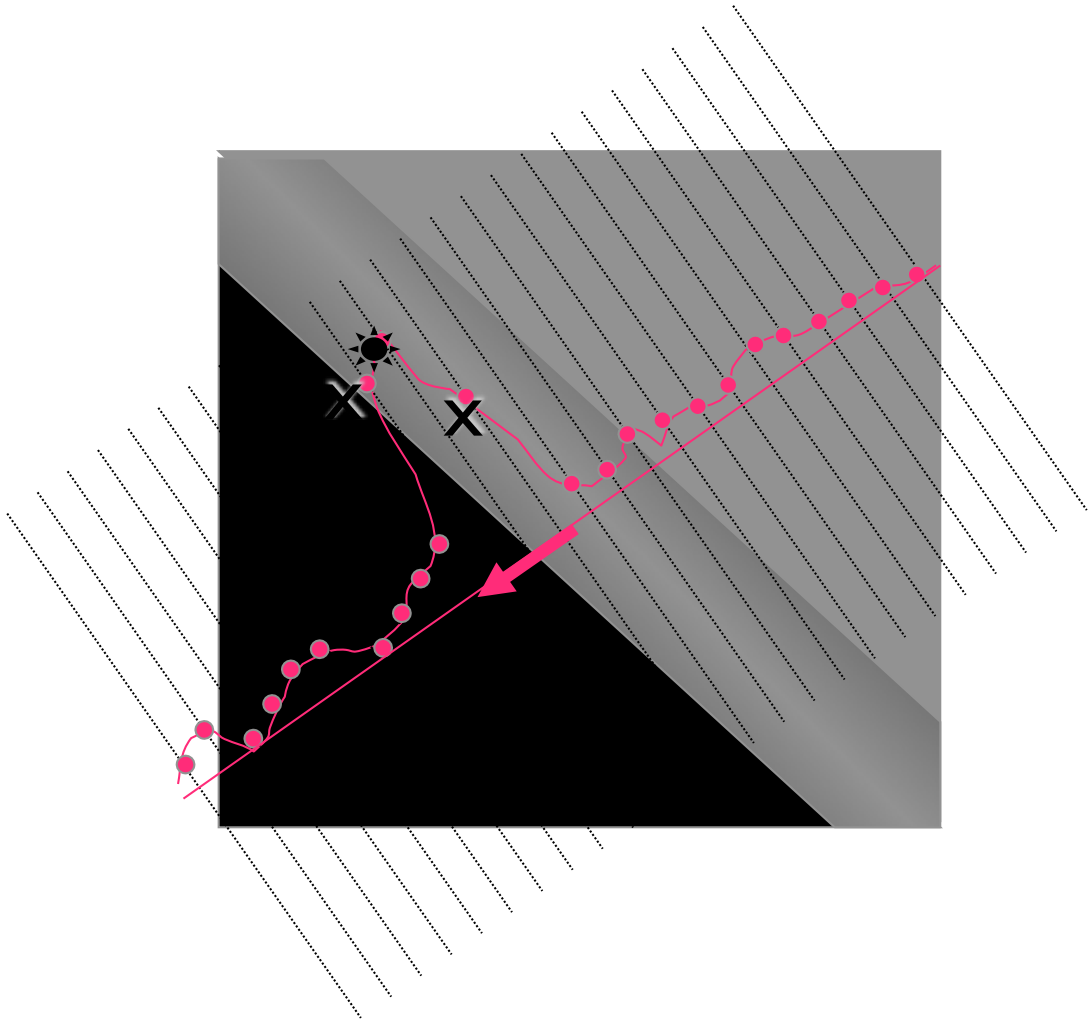
   If {$E_s(i,j)$ is smaller than at least one of its neigh. along d}

   1. $I_N(i,j)=0$

      Otherwise, $I_N(i,j)= E_s(i,j)$

The output is the thinned edge image I_N

Graphical Interpretation



Thresholding

Edges are found by thresholding the output of NONMAX_SUPRESSION

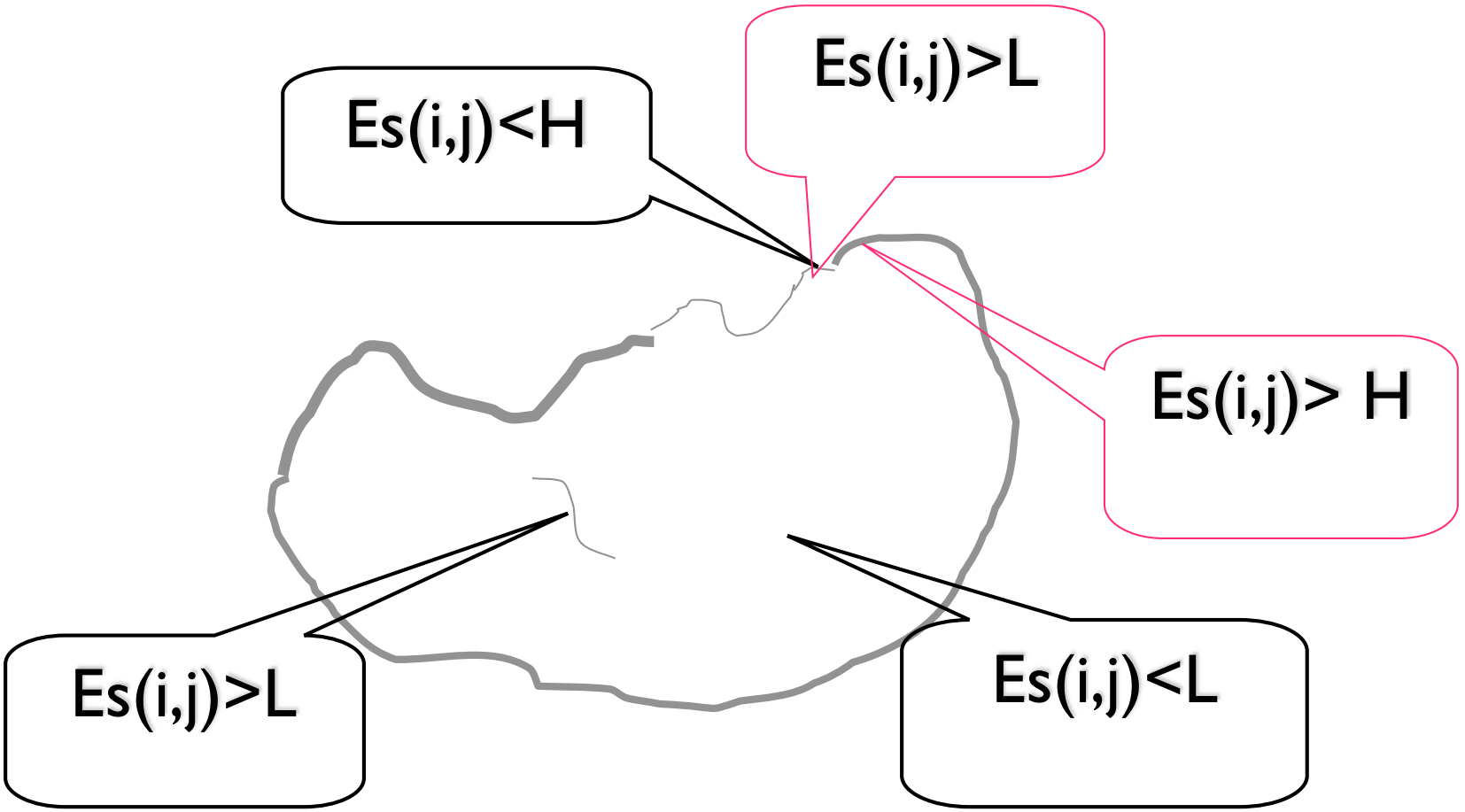
If the threshold is too high:

- Very few (none) edges
  - High MISDETECTIONS, many gaps

If the threshold is too low:

- Too many (all pixels) edges
  - High FALSE POSITIVES, many extra edges

SOLUTION:Hysteresis Thresholding



Strong edges reinforce adjacent weak edges

HYSTERESIS_THRESH

Inputs:

I_N (output of NONMAX_SUPRESSION),
E_o (output of CANNY_ENHANCER),
thresholds L and H.

For all pixels in I_N and scanning in a fixed order:

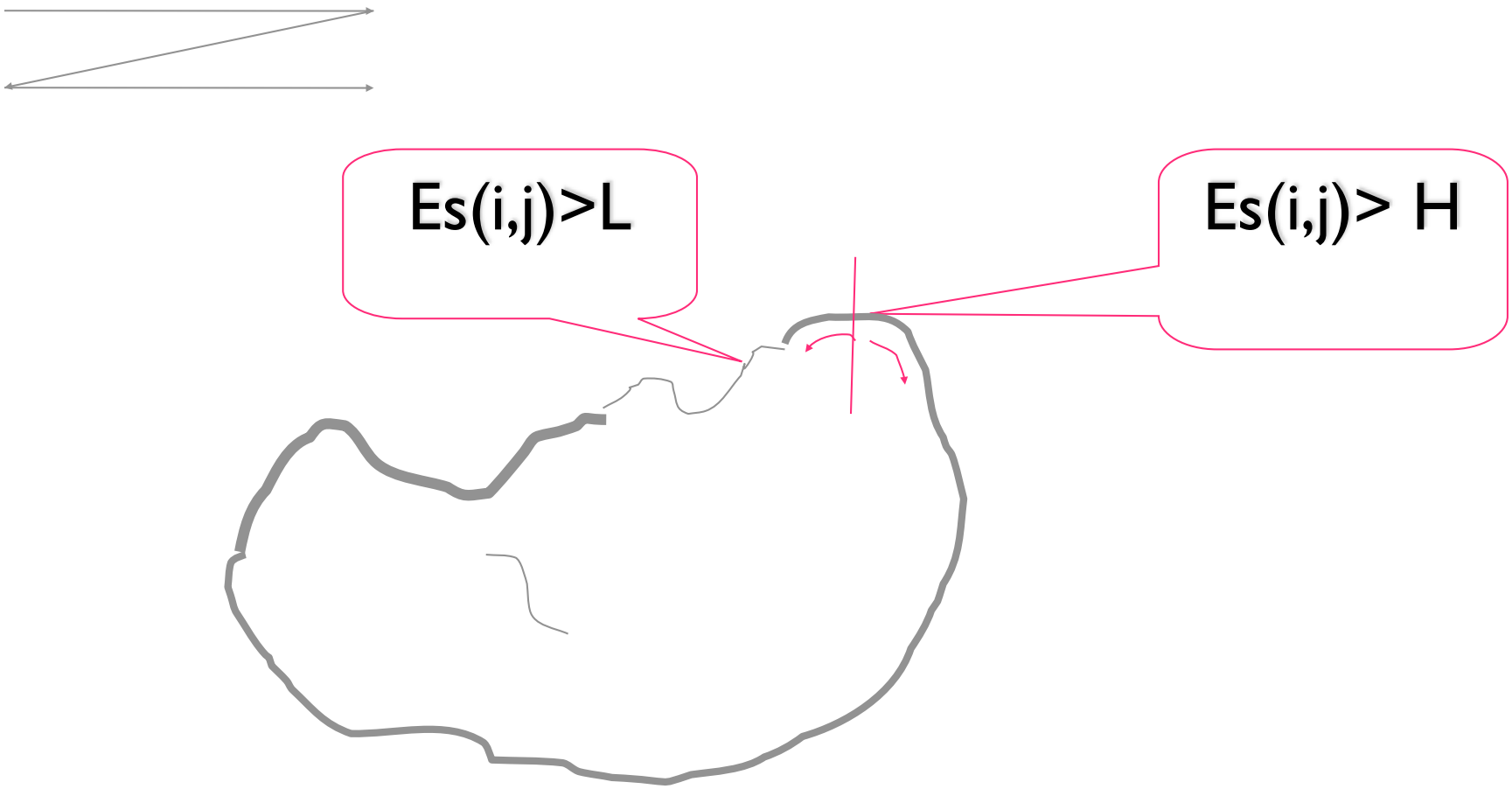
Locate the next unvisited pixel s.t. $I_N(i,j)>H$

Starting from $I_N(i,j)$, follow the chains of connected local maxima, in both directions perpendicular to the edge normal, as long as $I_N>L$.

Mark all visited points, and save the location of the contour points.

Output: a set of lists describing the contours.

Hysteresis Thresholding

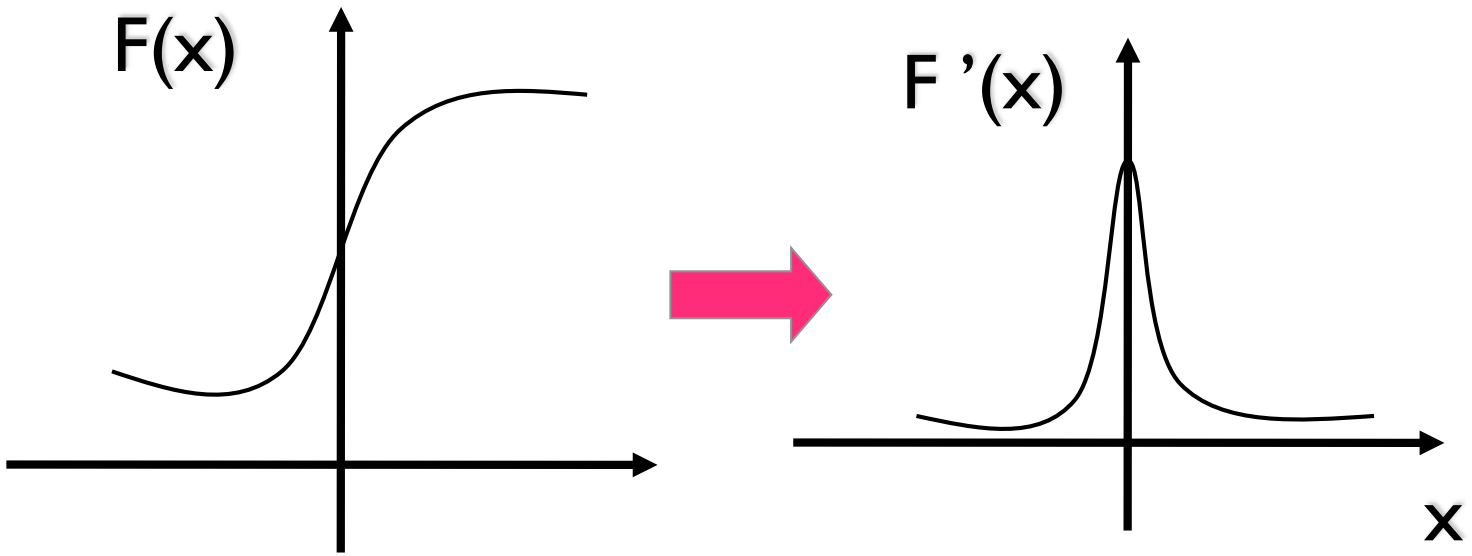


Other Edge Detectors

(2nd order derivative filters)

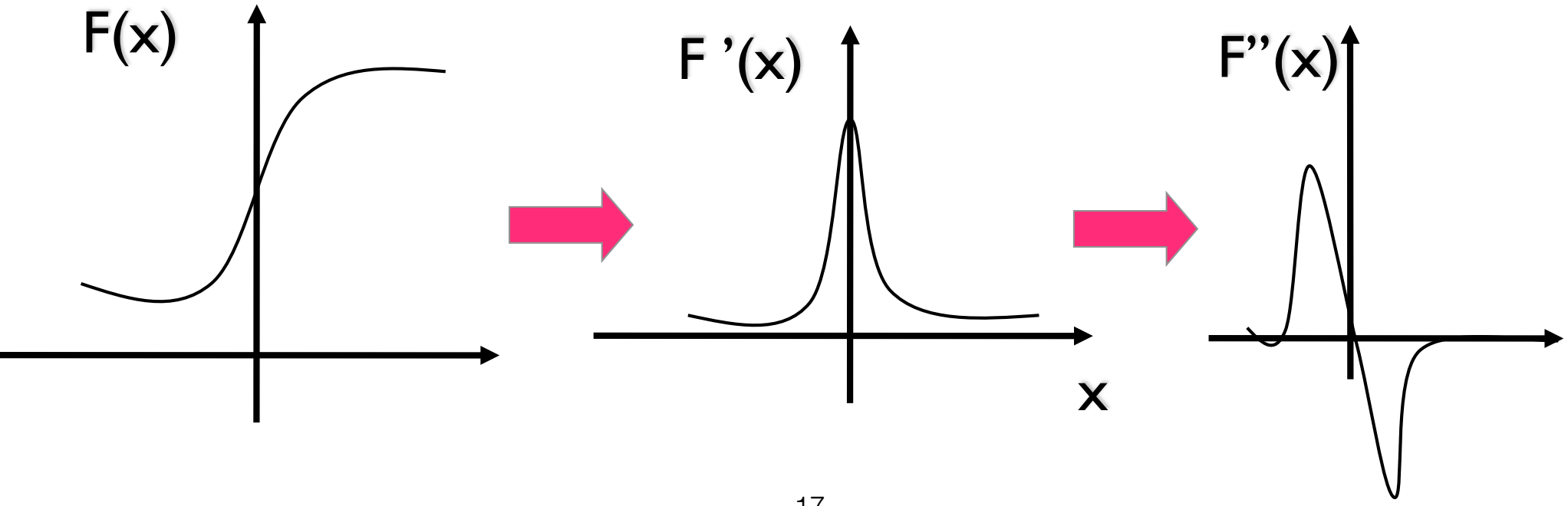
First-order derivative filters (1D)

Sharp changes in gray level of the input image correspond to “peaks” of the first-derivative of the input signal.



Second-order derivative filters (1D)

Peaks of the first-derivative of the input signal, correspond to “zero-crossings” of the second-derivative of the input signal.



NOTE:

F”(x)=0 is not enough!

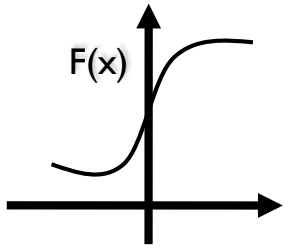
F’(x) = c has F”(x) = 0, but there is no edge

The second-derivative must change sign, -- i.e. from (+) to (-) or from (-) to (+)

The sign transition depends on the intensity change of the image – i.e. from dark to bright or vice versa.

Edge Detection (2D)

1D

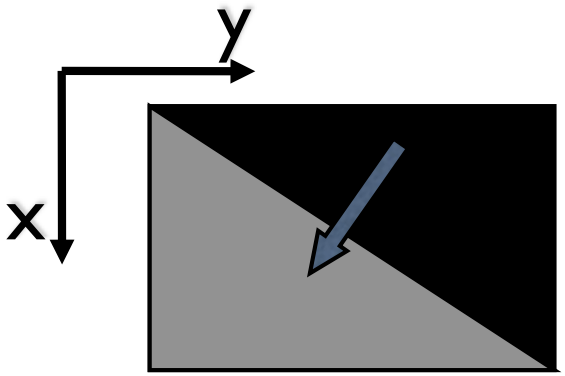


$I(x)$

$$\left| \frac{dI(x)}{dx} \right| > Th$$

$$\frac{d^2I(x)}{dx^2} = 0$$

2D



$I(x,y)$

$$|\nabla I(x,y)| = (I_x^2(x,y) + I_y^2(x,y))^{1/2} > Th$$

$$\tan\theta = I_x(x,y) / I_y(x,y)$$

$$\nabla^2 I(x,y) = I_{x\,x}(x,y) + I_{yy}(x,y) = 0$$

Laplacian

Notes about the Laplacian:

$\nabla^2 I(x,y)$ is a SCALAR

↑ Can be found using a SINGLE mask

↓ Orientation information is lost

$\nabla^2 I(x,y)$ is the sum of SECOND-order derivatives

But taking derivatives increases noise

Very noise sensitive!

It is always combined with a smoothing operation

LOG Filter

First smooth (Gaussian filter),

Then, find zero-crossings (Laplacian filter):

$$O(x,y) = \nabla^2(I(x,y) * G(x,y))$$

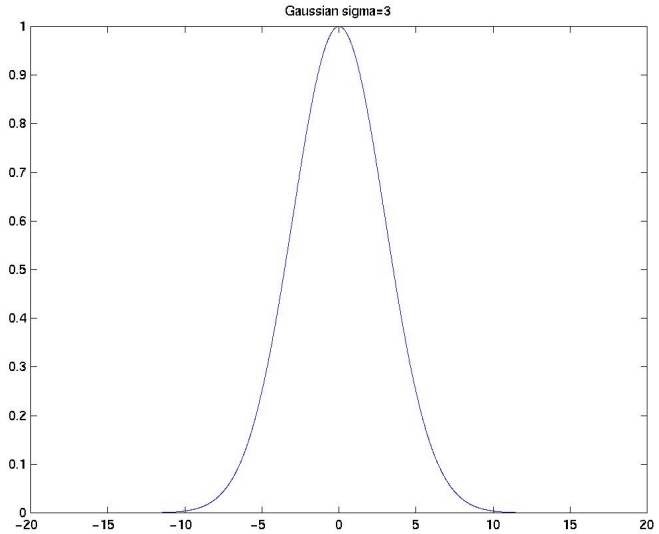
Using linearity:

$$O(x,y) = \nabla^2 G(x,y) * I(x,y)$$

This filter is called: "Laplacian of the Gaussian" (LOG)

1D Gaussian

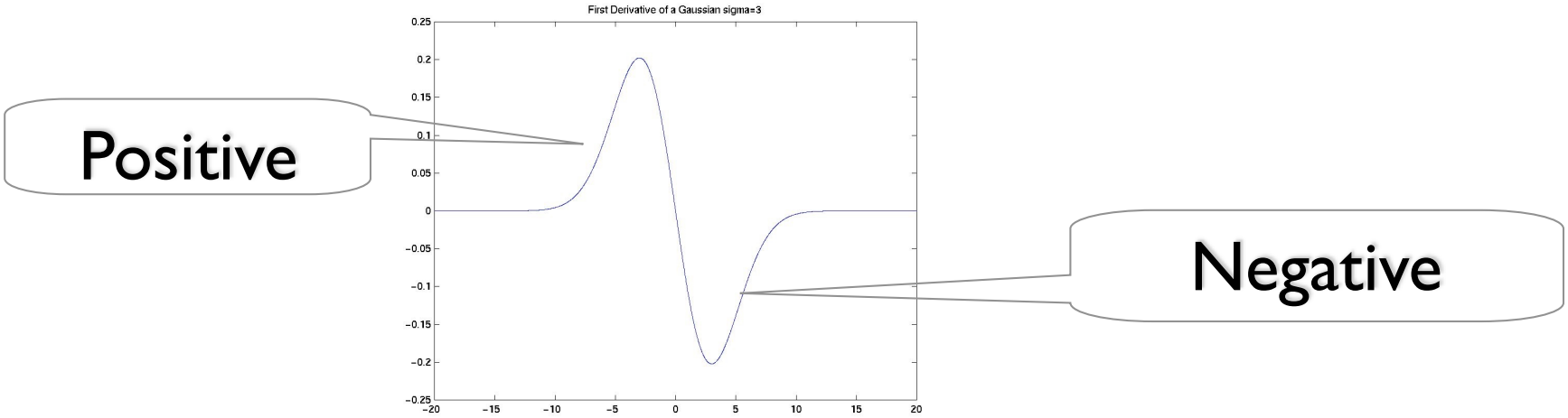
$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$



$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

First Derivative of a Gaussian

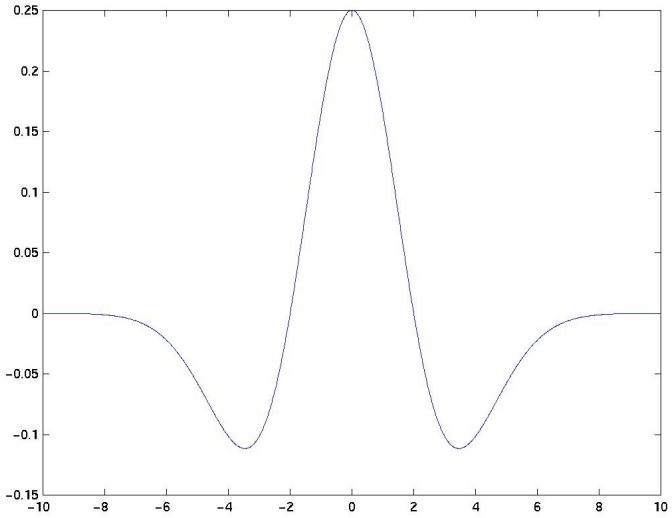
$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$



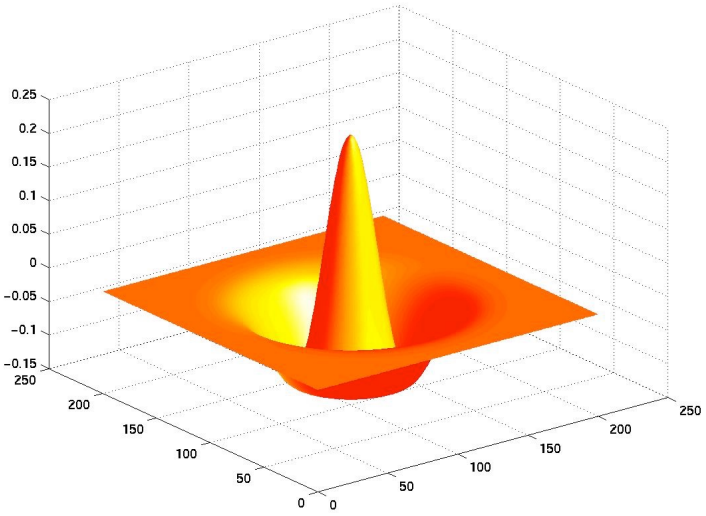
As a mask, it is also computing a difference (derivative)

Second Derivative of a Gaussian

$$g''(x) = \left(\frac{x^2}{\sigma^3} - \frac{1}{\sigma}\right) e^{-\frac{x^2}{2\sigma^2}}$$



2D



"Mexican Hat"

Modern Edge Detection

P. Dollar and C. Zitnick:

"Structured Forests for Fast Edge Detection," ICCV 2013.

Code: http://research.microsoft.com/en-us/downloads/389109f6-b4e8-404c-84bf-239f7cbf4e3d/default.aspx

Let's look at Canny's output

Task: Find the cows in the image

Canny Edges

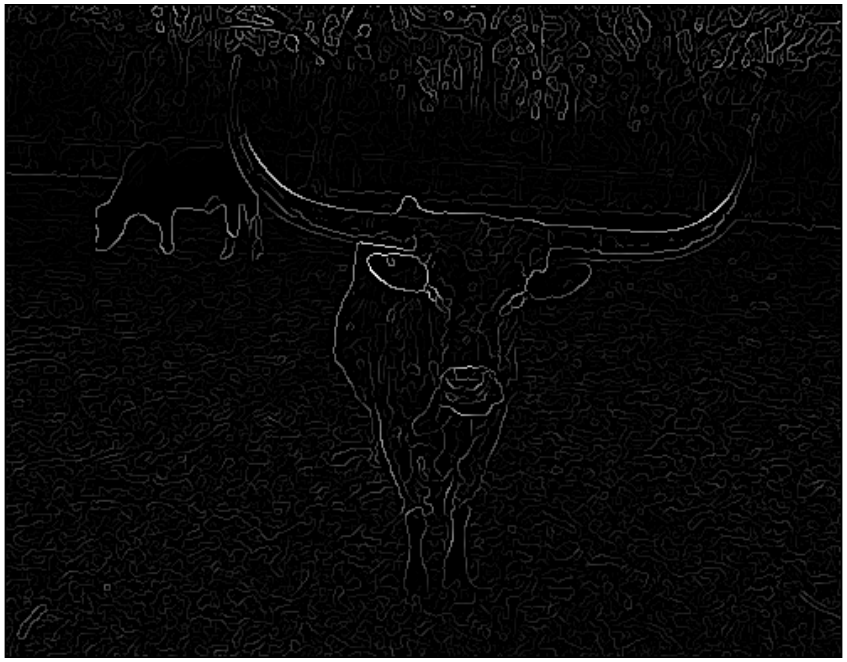


Image Gradients + NMS

Canny's Edges

Canny Edges

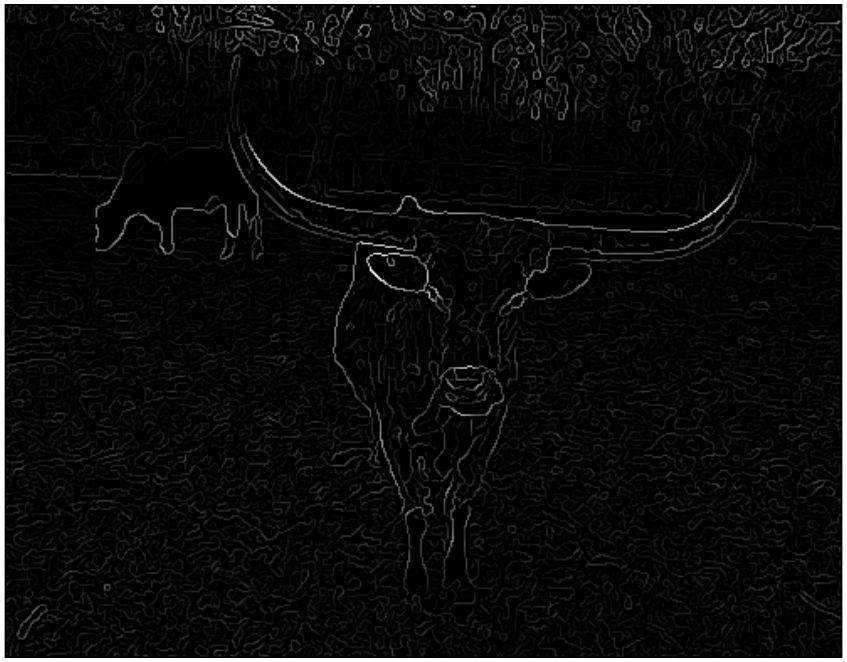


Image Gradients + NMS

Canny's Edges

Canny Edges

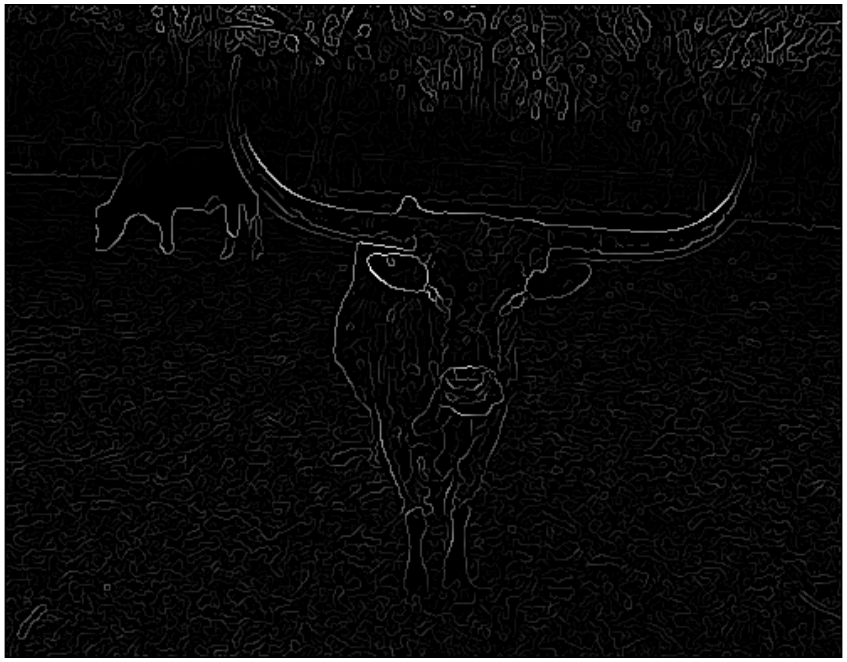


Image Gradients + NMS

Canny's Edges

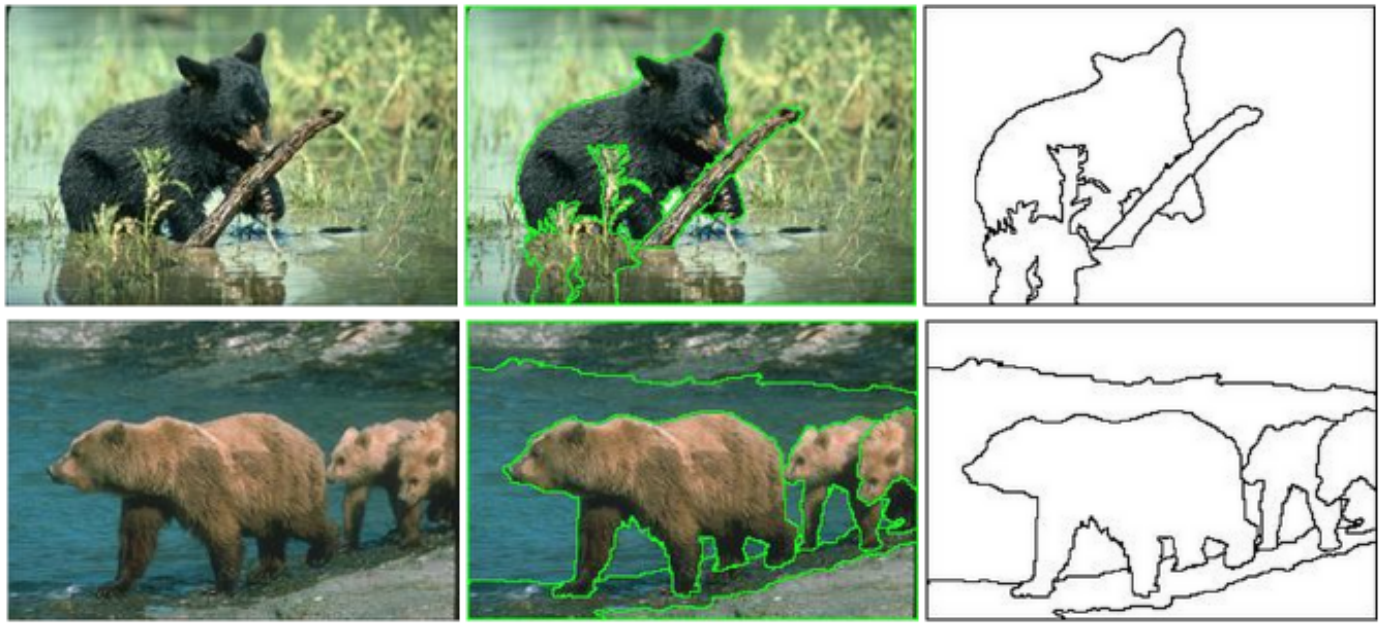
Many distractors, missing edges!

Can we do better?

Imagine … that someone goes and ANNOTATES which edges matter

Annotate

Some people have done that!



The Berkeley Segmentation Dataset and Benchmark

by D. Martin, C. Fowlkes, D. Tal and J. Malik

Using Annotations

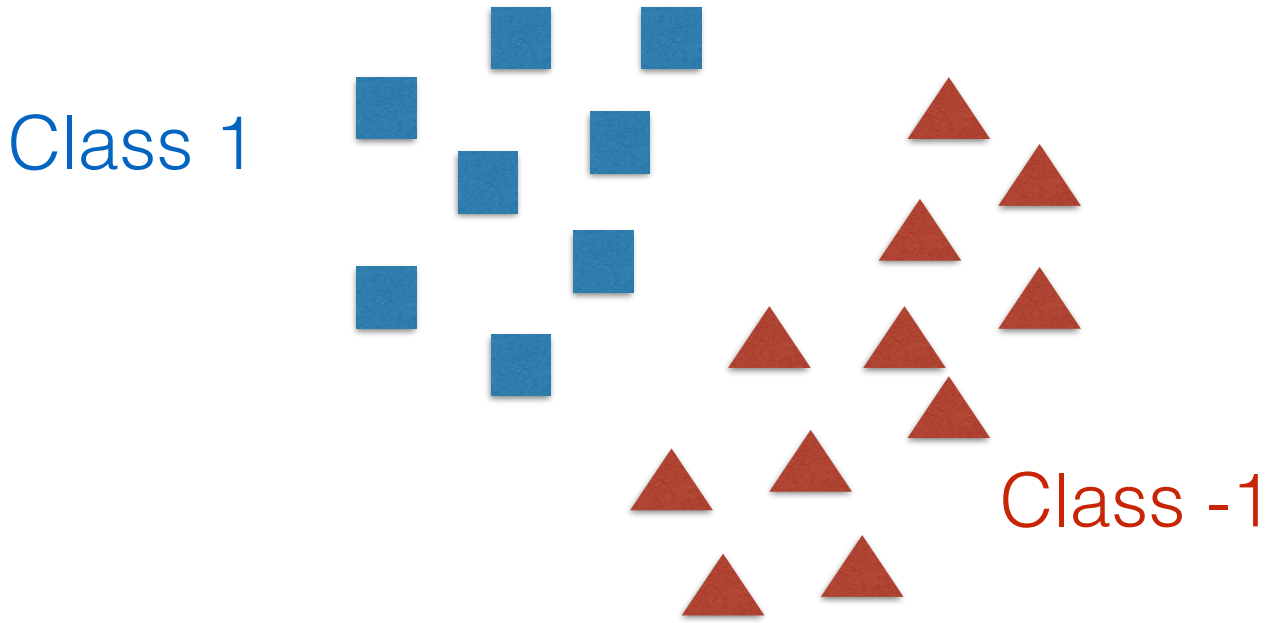
How can we use annotations to improve (task oriented) edge detection?

We can use Machine Learning!

Very over-simplified explanation

Supervised Classification Problem:

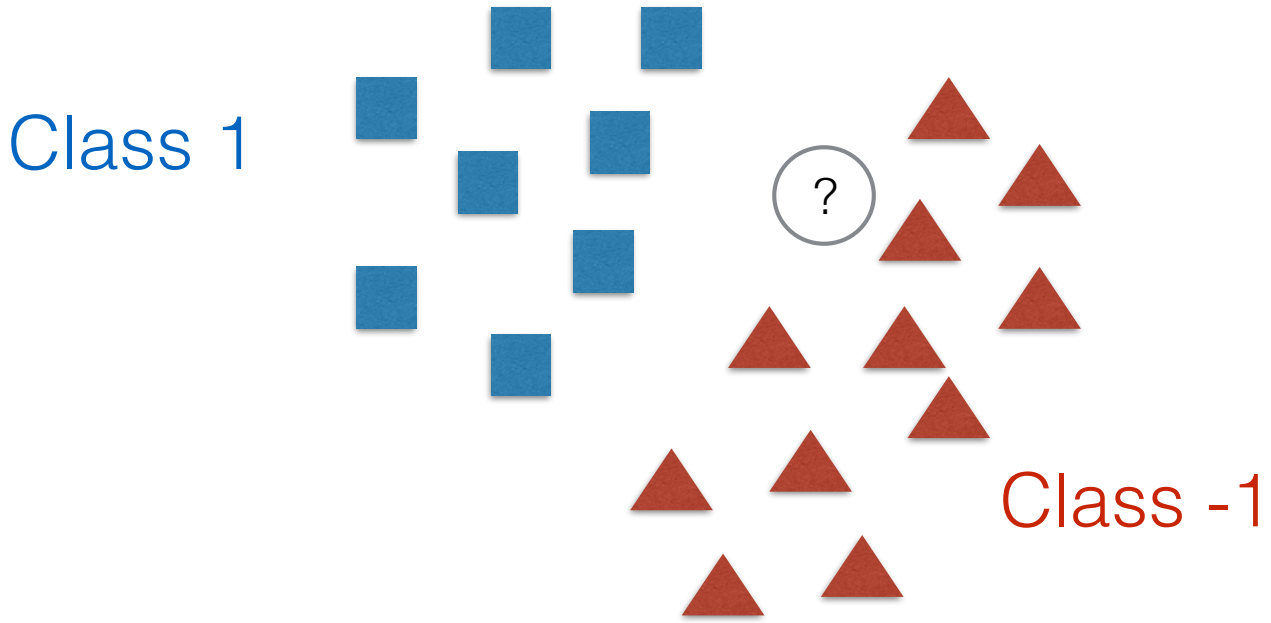
Samples are represented by “features”



We have “training” labeled data from 2 classes.

Very over-simplified explanation

Supervised Classification Problem:

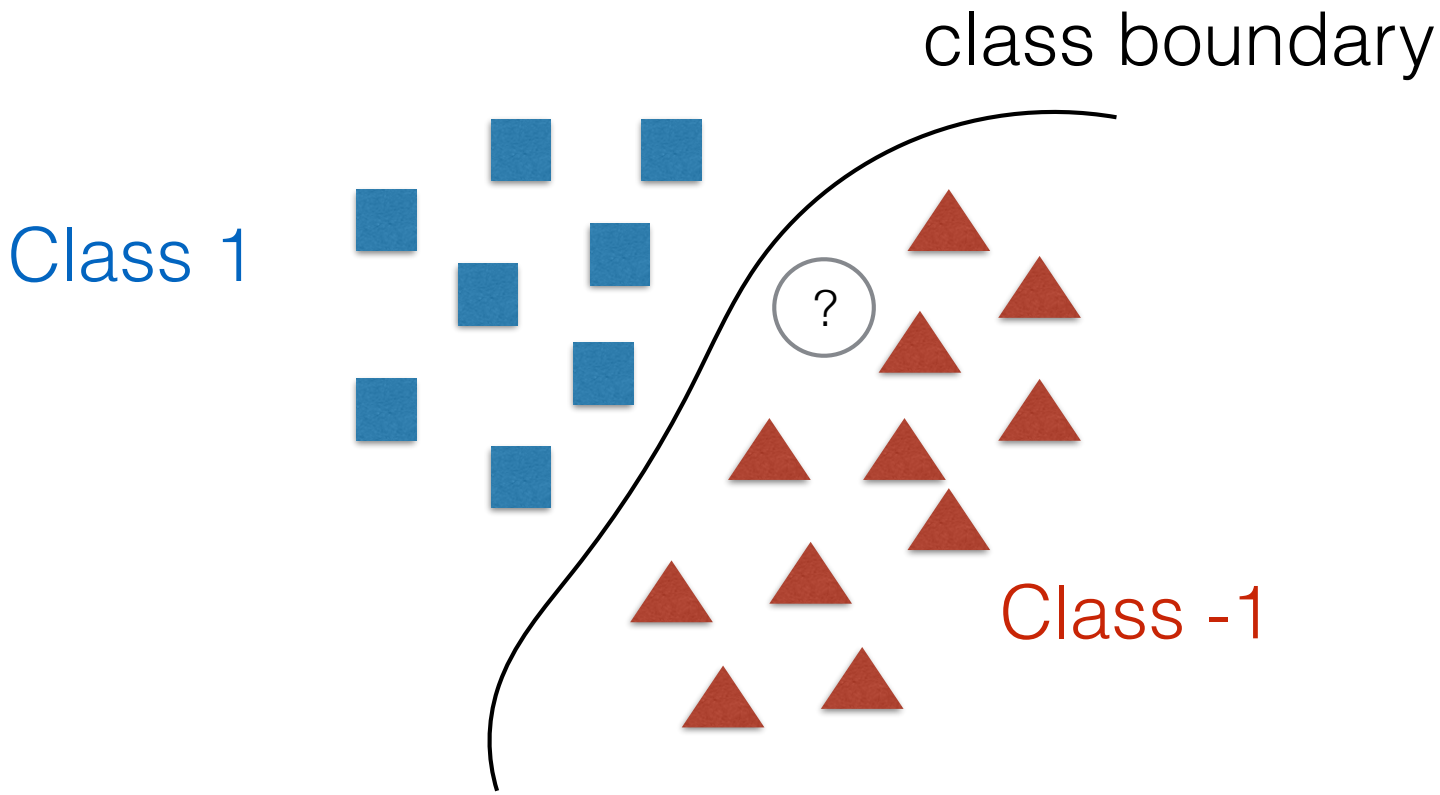


We have “training” labeled data from 2 classes.

Given a new sample, we want to assign to it a label.

Very over-simplified explanation

Supervised Classification Problem:

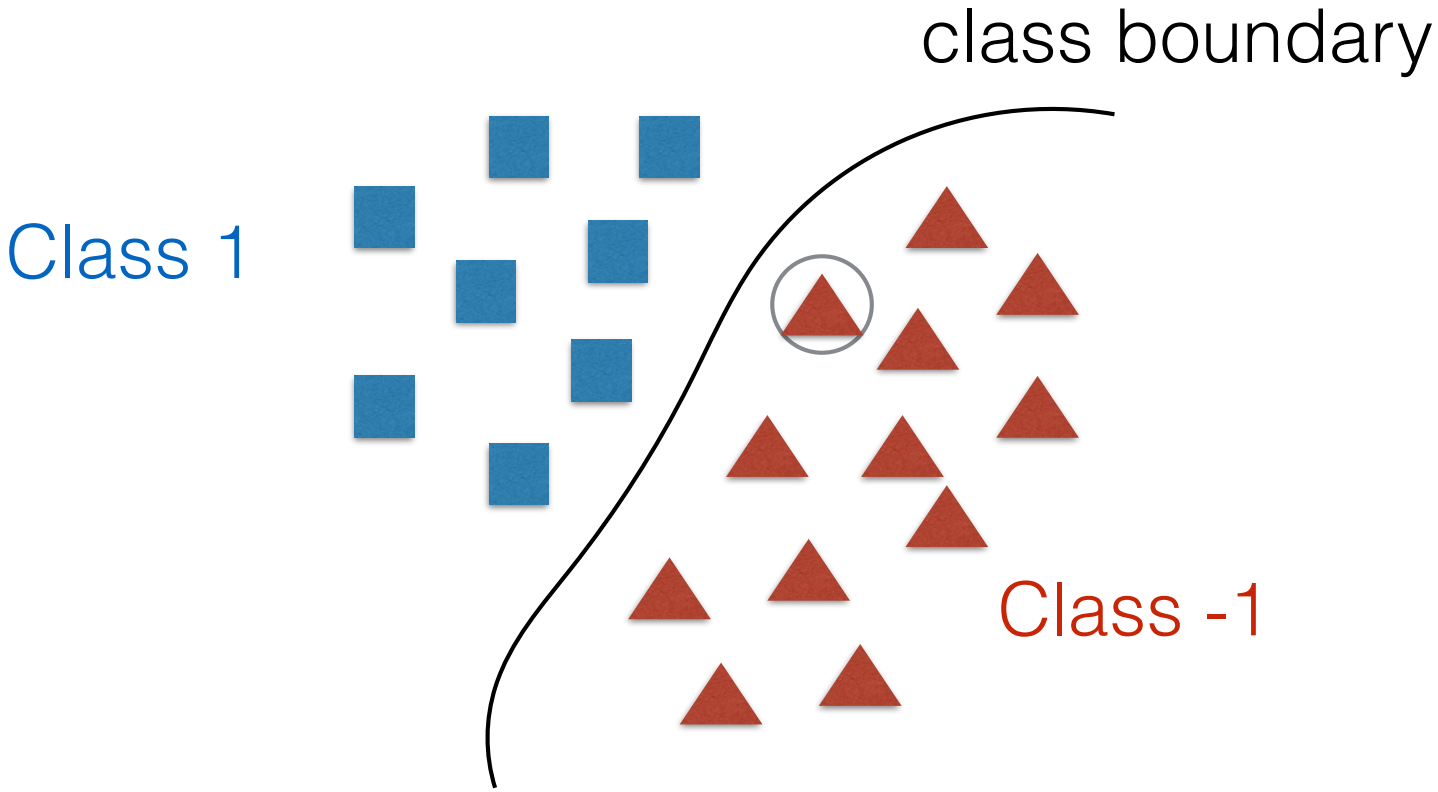


We have “training” labeled data from 2 classes.

Given a new sample, we want to assign to it a label.

Very over-simplified explanation

Supervised Classification Problem



We have “training” labeled data from 2 classes.

Given a new sample, we want to assign to it a label.

Supervised Classification

There are many strategies to learn classifiers:

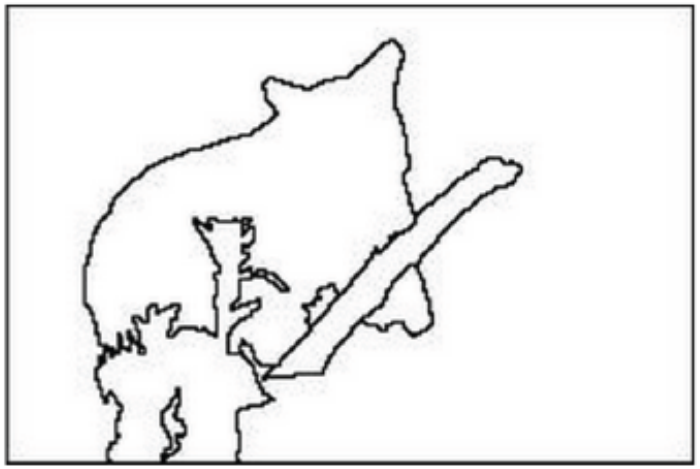
- Nearest neighbors
- Support Vector Machine
- Classification Trees
- Classification Forests
- Neural networks (deep learning)

Classifiers provide:

- a class “label” for new samples
- a “score” of how well the label fits the new sample

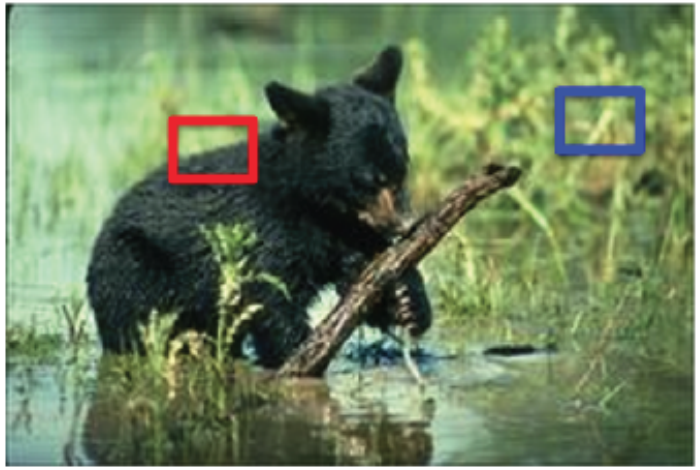
Training an Edge Detector

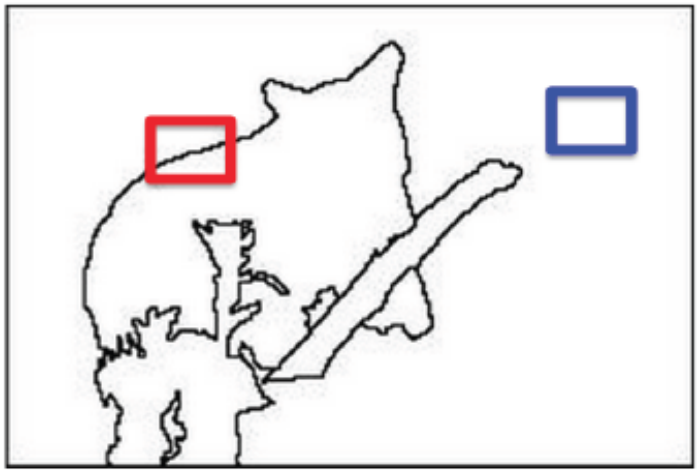
image



annotation

Training an Edge Detector





Labeled Samples!

Recipe for Training an Edge Detector

- Extract LOTS of image patches
- Assign labels to the samples using annotations
- Represent each patch using features (measurements) of the patch
  - Simplest possibility would be to vectorize the patch:

matrix $\mathbf{P}$

$$\rightarrow \quad \mathbf{x} = \mathbf{P}(:)$$

Recipe for Training an Edge Detector

- Extract LOTS of image patches
- Assign labels to the samples using annotations
- Represent each patch using features (measurements) of the patch
  - Simplest possibility would be to vectorize the patch
  - Better: extract meaningful features such as gradients, color histogram, etc

matrix $\mathbf{P}$

compute gradients

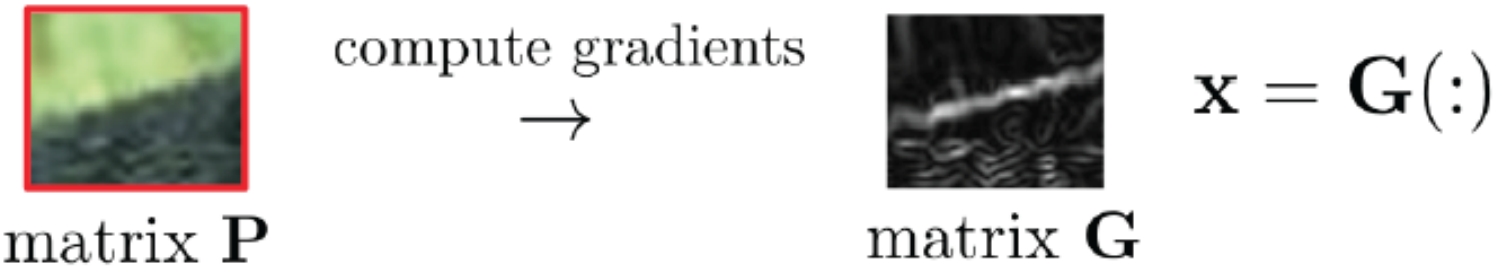
$\rightarrow$

matrix $\mathbf{G}$

$$\mathbf{x} = \mathbf{G}(:)$$

Recipe for Training an Edge Detector

- Extract LOTS of image patches
- Assign labels to the samples using annotations
- Represent each patch using features (measurements) of the patch
  - Simplest possibility would be to vectorize the patch
  - Better: extract meaningful features such as gradients, color histogram, etc
- Train the classifier

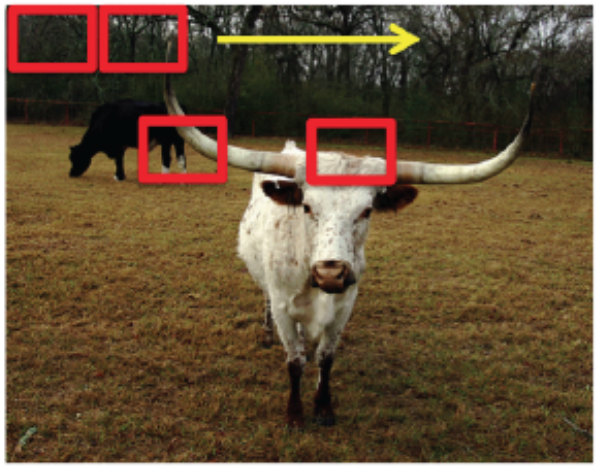


Using the Classifier

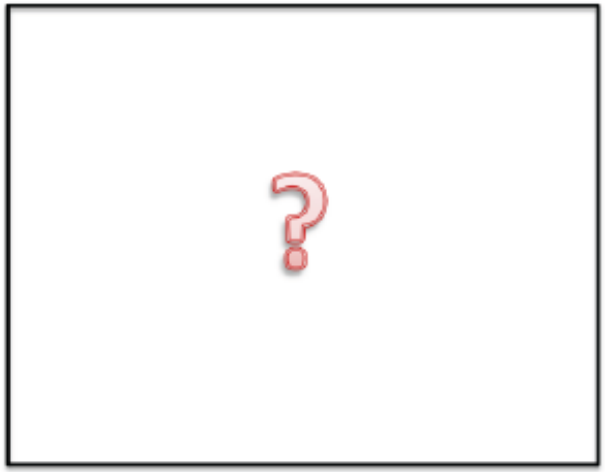
image

prediction

Using the Classifier

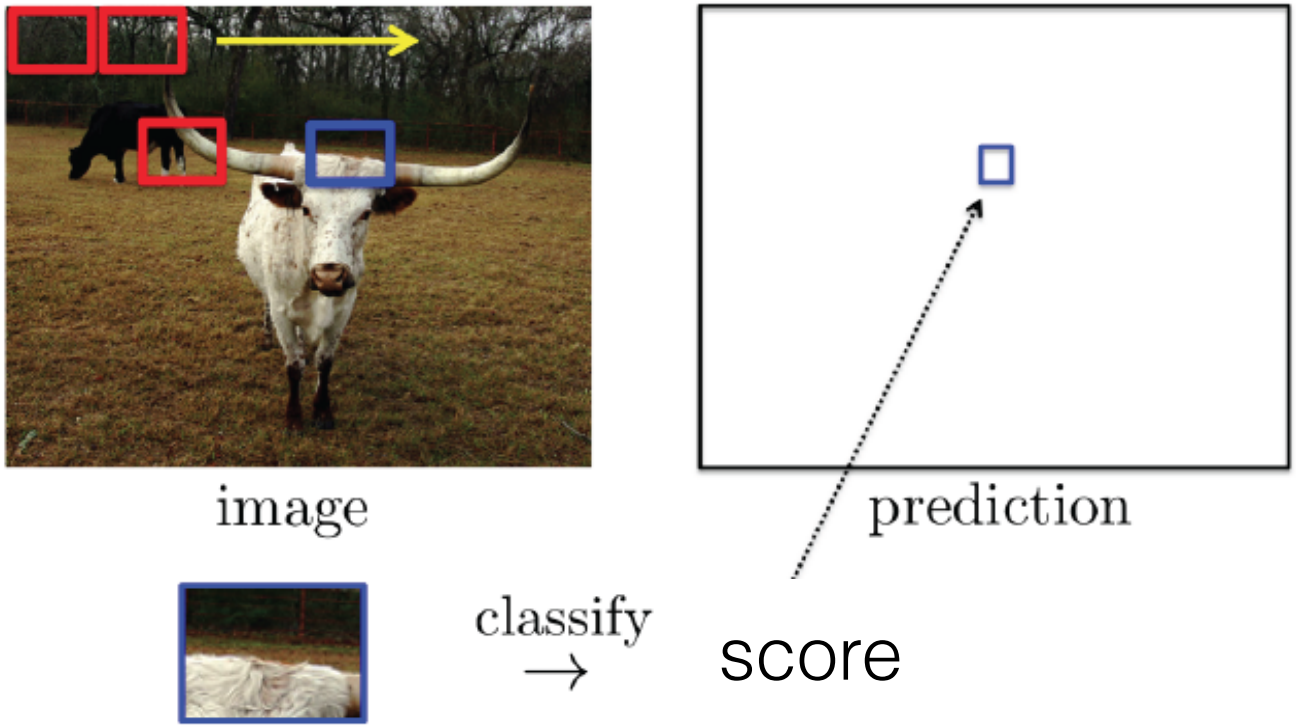


image



prediction

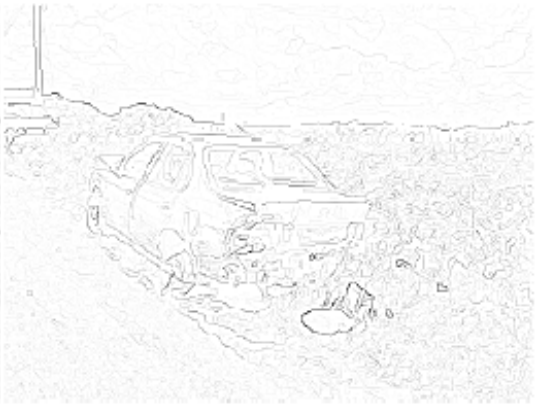
Using the Classifier



Canny vs Structured Edge Detector

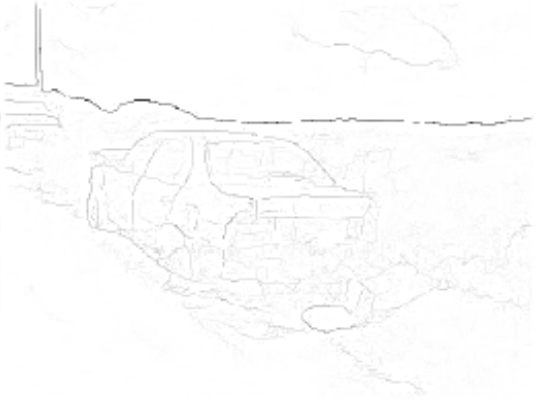
image

image gradients



gradients + NMS

"edgeness score"



score + NMS

Canny vs Structured Edge Detector

image

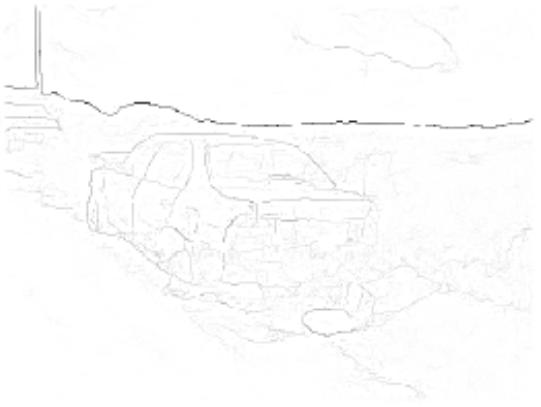
image gradients

gradients + NMS

image gradient

"edgeness" score

"edgeness" score



score + NMS

Canny vs Structured Edge Detector

image

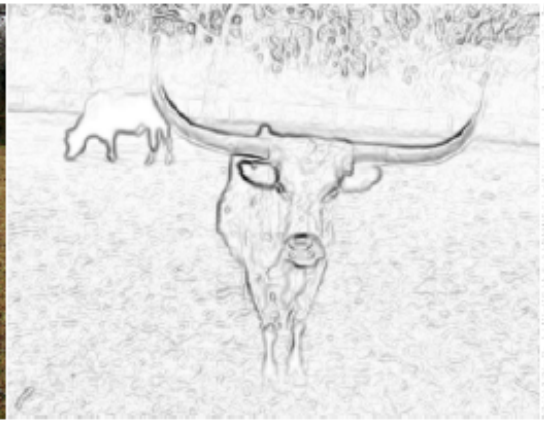
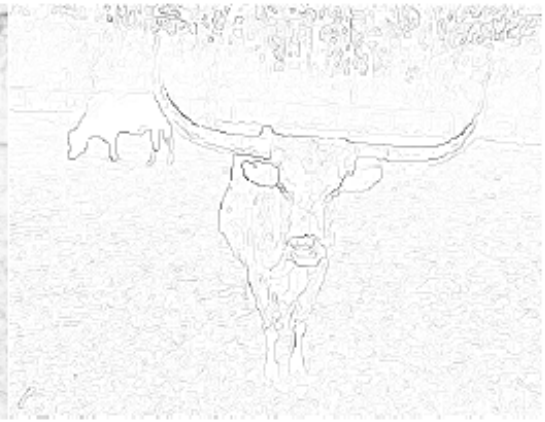
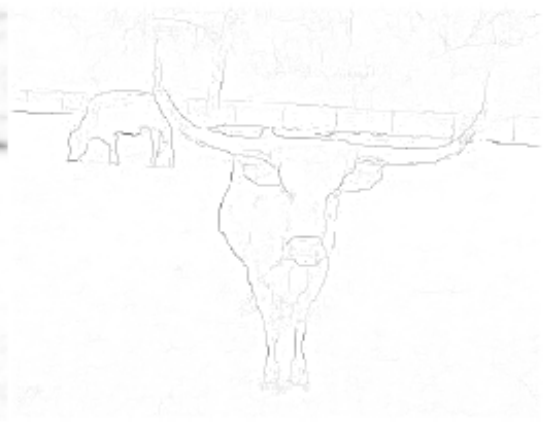


image gradients



gradients + NMS

"edgeness" score



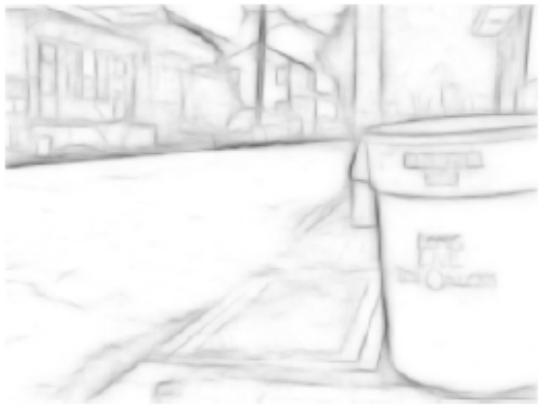
score + NMS

Canny vs Structured Edge Detector

image

image gradients

gradients + NMS



"edgeness" score

score + NMS

Canny vs Structured Edge Detector

image

image gradients

gradients + NMS

image gradient

"edgeness" score

"edgeness" score

score + NMS

Evaluating Results

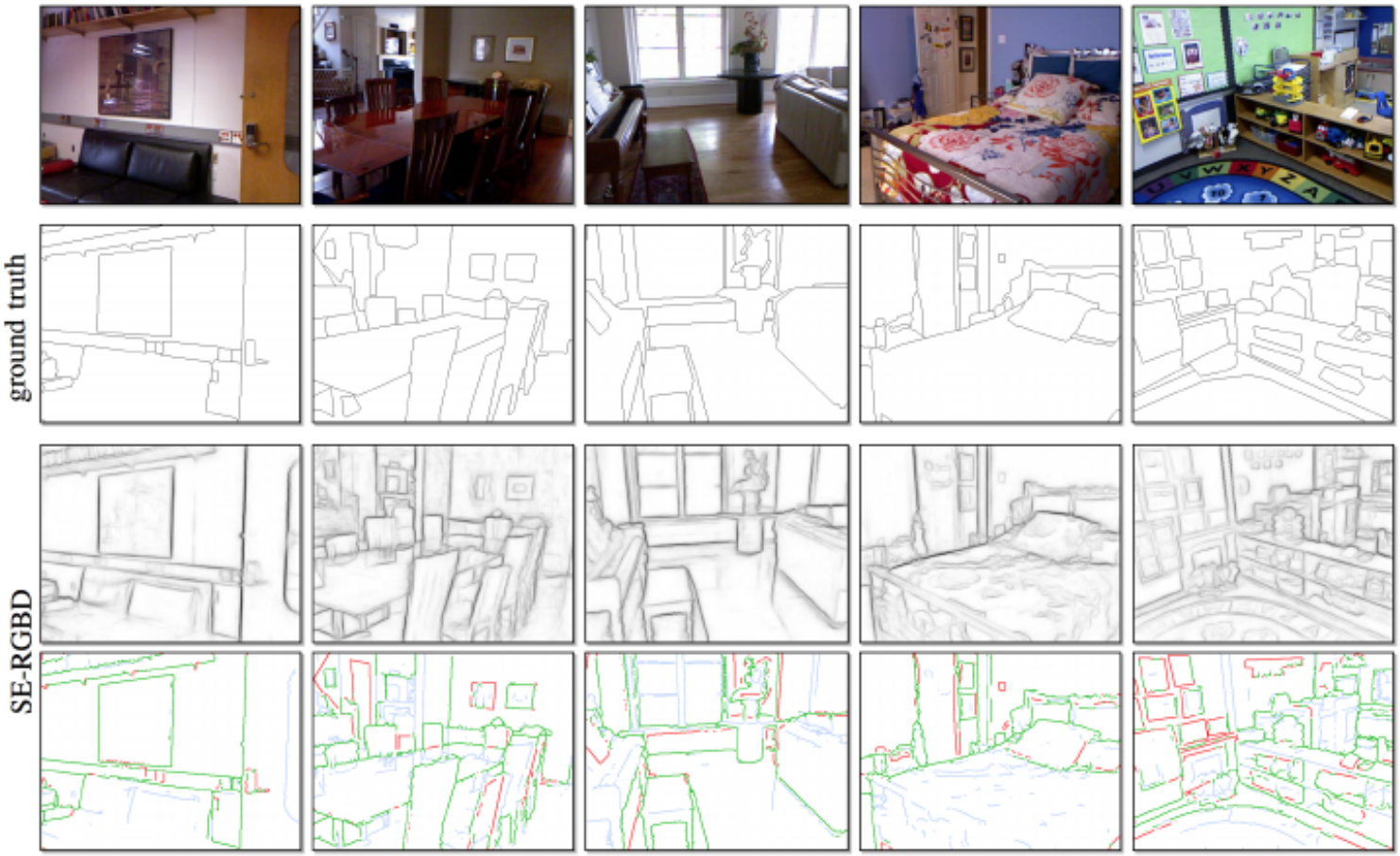
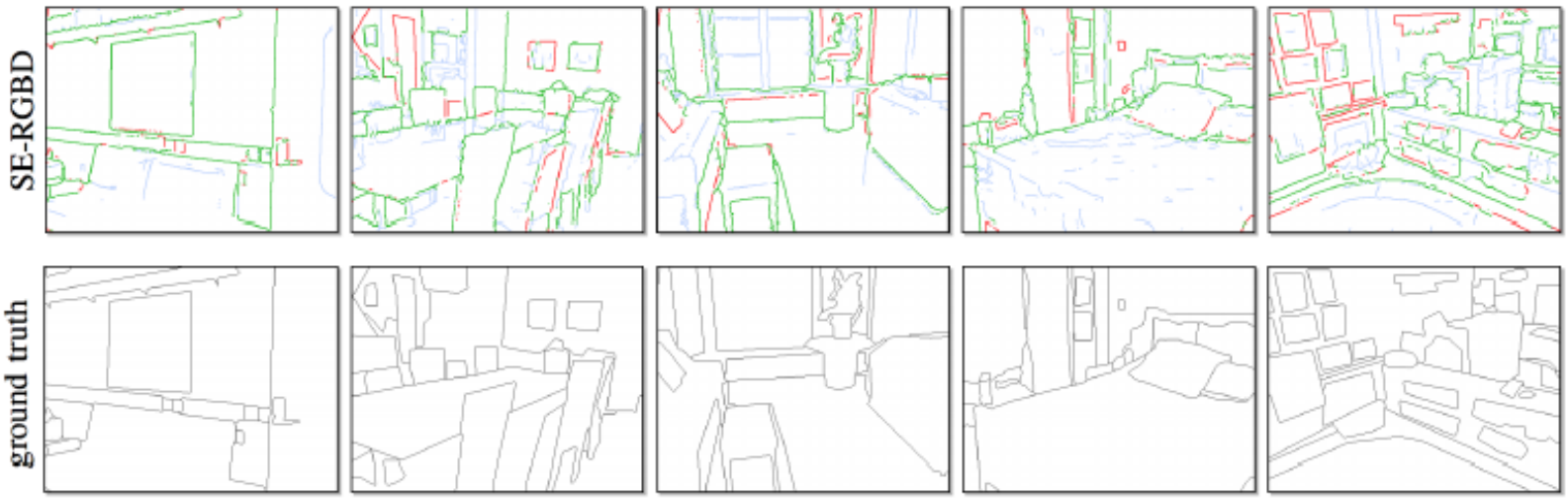


Figure: green=correct, blue=wrong, red=missing, green+blue=output edges

Evaluation: Recall & Precision

- **Recall:** How many of all **annotated** edges we got correct (best is 1)
- **Precision** How many of all **output** edges we got correct (best is 1)

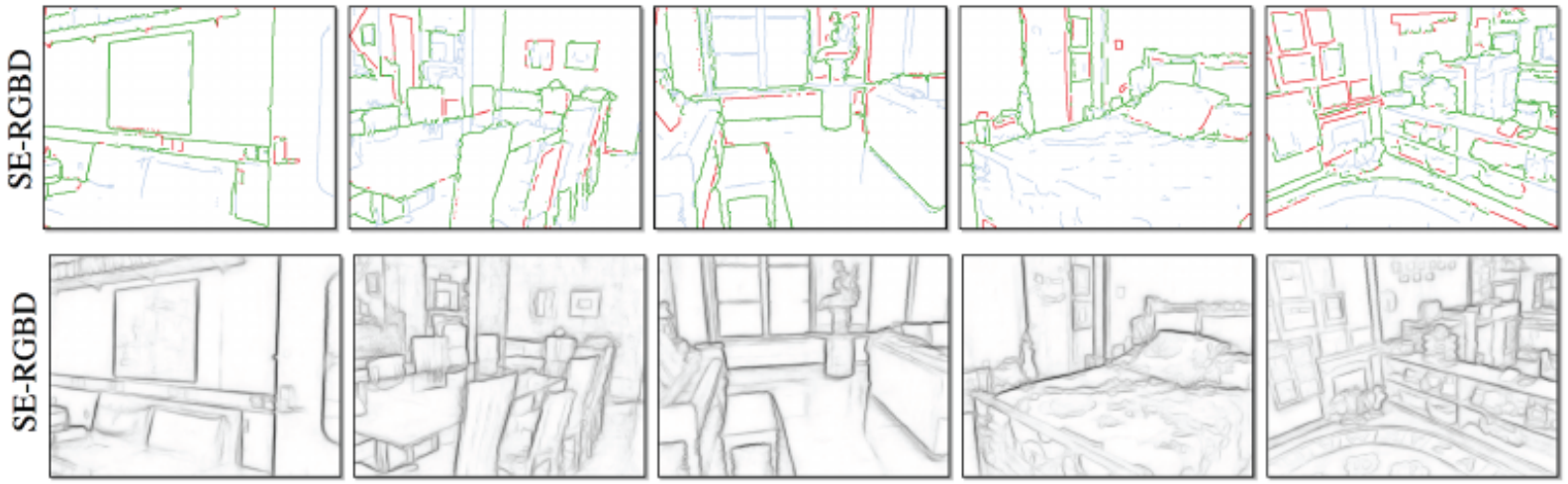
$$\textbf{Recall} = \frac{\#\text{ of green (correct edges)}}{\#\text{ of all edges in ground-truth (second picture)}}$$



Evaluation: Recall & Precision

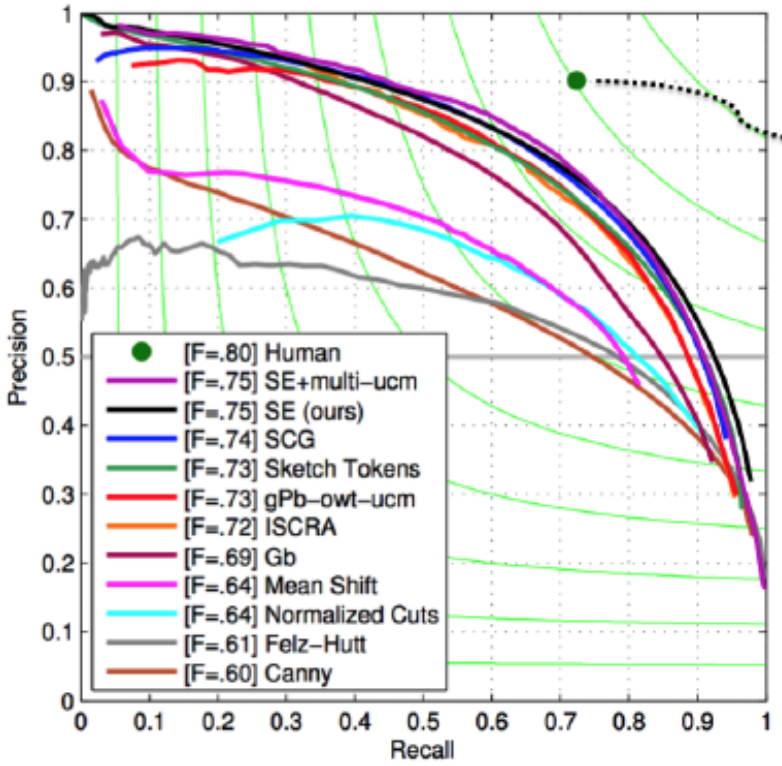
- **Recall:** How many of all **annotated** edges we got correct (best is 1)
- **Precision** How many of all **output** edges we got correct (best is 1)

$$\textbf{Precision} = \frac{\#\text{ of green (correct edges)}}{\#\text{ of all edges in output (second picture)}}$$



Evaluation

- **Recall:** How many of all **annotated** edges we got correct (best is 1)
- **Precision** How many of all **output** edges we got correct (best is 1)



human agreement

Precision-recall Curve

Point Features
Corners

Corner Features

Places where TWO strong edges meet.

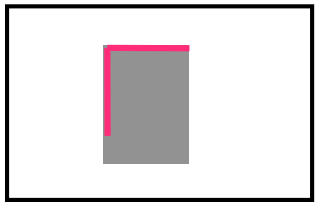
They can be used for:

- Object tracking
- 3D triangulation (stereo)
- Object recognition

Detection of Corner Features

Need two strong edges:

Example:



Create the following matrix:

$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}$$

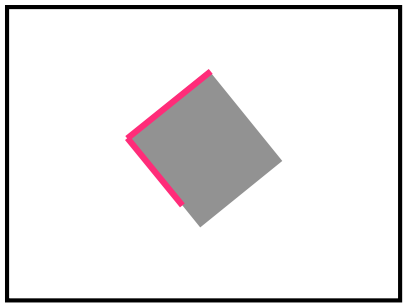
Either E_x or E_y but not both are large in a neighborhood of corner

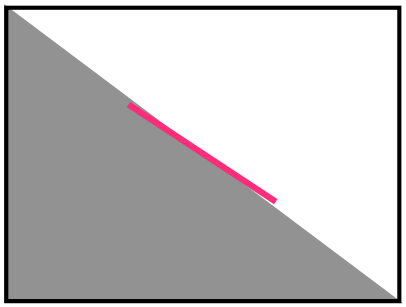
$$\Rightarrow \quad C = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

If $\min(\lambda_1, \lambda_2) > T$
There is a corner!

Detection of Corner Features

What if the corner is not aligned with the image coordinate system?





$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}$$

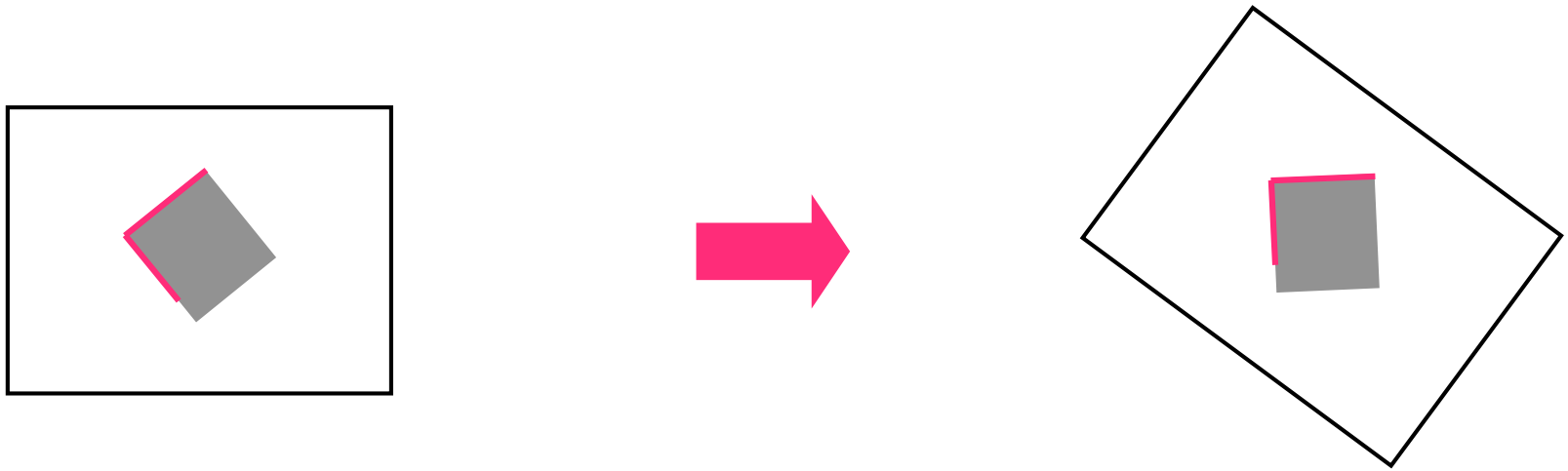
But this is also true for a slanted edge!

Both, E_x or E_y are large in the neighborhood of the corner

Detection of Corner Features

Solution:

"Rotate" the corner to align it with the image coordinate system!



$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix} \qquad C = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

Detection of Corner Features

How do we do this rotation?

Since C is symmetric, it can be diagonalized;

the diagonalization is done by the rotation we need!

Review:Matrix Eigenvalues & Eigenvectors

Let

A be a SQUARE matrix

v be a column vector

λ an scalar

If $A.v = \lambda v$

Assuming the non trivial solution ($v \neq 0$)

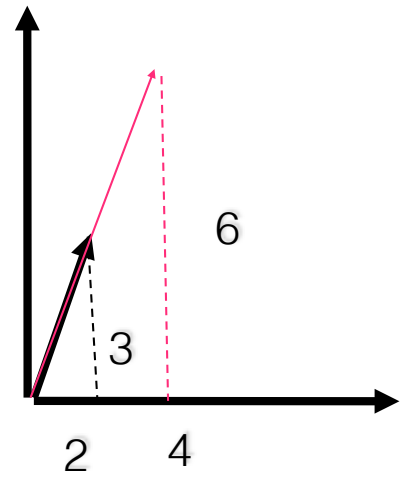
v is an EIGENVECTOR of A

λ is an EIGENVALUE of A

Example:

$$A = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix}$$

$$v = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$



$$Av = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix} = 2 \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

v is an eigenvector with eigenvalue 2

Finding Eigenvalues

$$Av = \lambda v$$

$$Av - \lambda v = 0$$

$$(A - \lambda I)v = 0$$

(A - λ I) v = 0 is an HOMOGENEOUS LINEAR system of equations

Finding Eigenvalues

(A - λ I) v = 0 is an HOMOGENEOUS LINEAR system of equations

v is an EV if it is not the TRIVIAL solution v=0

$$det(A - \lambda I) = 0$$

Example:

$$A = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix}$$

$$\det(A - \lambda I) = \det \begin{vmatrix} 5-\lambda & -2 \\ 6 & -2-\lambda \end{vmatrix} = 0$$

$$(5-\lambda)(-2-\lambda) - (6)(-2) = 0$$

$$\lambda^2 - 3\lambda + 2 = 0 \;\Rightarrow \lambda = \frac{3 \pm \sqrt{9-8}}{2}$$

$$\lambda_1 = 1, \lambda_2 = 2$$

Eigenvector for λ = 1

$$\begin{aligned} (5-1)v_{11} - 2v_{12} &= 0 \\ 6v_{11} + (-2-1)v_{12} &= 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 4v_{11} - 2v_{12} &= 0 \\ 6v_{11} - 3v_{12} &= 0 \end{aligned}$$

$$\Rightarrow 2v_{11} = v_{12} \Rightarrow \quad v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Eigenvector for $\lambda = 2$

$$\begin{aligned} (5-2)v_{21} - 2v_{22} &= 0 \\ 6v_{21} + (-2-2)v_{22} &= 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 3v_{21} - 2v_{22} &= 0 \\ 6v_{21} - 4v_{22} &= 0 \end{aligned}$$

$$\Rightarrow 3v_{21} = 2v_{22} \quad \Rightarrow \quad v_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Properties:

Trace(A) = $\sum \lambda_i$

$$Trace(A) = Trace\begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix} = 5 - 2 = 3$$

$$\sum \lambda_i = 1 + 2 = 3$$

Properties:

det(A) = $\Pi\ \lambda_i$

$$\det(A) = \det\begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix} = 5(-2) - 6(-2) = 2$$

$$\Pi\lambda_i = 1 \cdot 2 = 2$$

Properties:

If there is a NULL eigenvalue, A is not invertible: det(A)=0

A and A^2 have the same eigenvectors.

If λ is and EV of A, λ^2 is an EV of A^2

Symmetric Matrix

$$A = A^T$$

With ev v_i and v_j and eval λ_i λ_j

v_i is an ev:

$$Av_i = \lambda_i v_i$$

Multiply both sides by v_j^T

$$\underbrace{v_j^T A} v_i = \lambda_i v_j^T v_i$$

A is symmetric:

$$v_j^T A = v_j^T A^T$$

v_j is an ev:

$$v_j^T A^T = \lambda_j v_j^T$$

$$\lambda_j v_j^T v_i = \lambda_i v_j^T v_i$$

Symmetric Matrix

$$A = A^T$$

With ev v_i and v_j and eval λ_i λ_j

$$\lambda_j v_j^T v_i = \lambda_i v_j^T v_i$$

$$(\lambda_j - \lambda_i) v_j^T v_i = 0 \quad \Rightarrow \quad v_i \perp v_j$$

The eigenvectors of $A = A^T$ are orthogonal to each other

Matrix Diagonalization:

Let A nxn with:

n DIFFERENT eigenvalues λ_1 λ_2 … λ_n

$v_1, v_2 \ldots v_n$ eigenvectors

$$Av_i = \lambda_i v_i$$

Let P nxn with columns $v_1, v_2, \ldots v_n$:

$$P = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{bmatrix}$$

Matrix Diagonalization

$$P = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \dots & v_n \\ | & | & & | \end{bmatrix}$$

$$AP = \begin{bmatrix} | & | & & | \\ Av_1 & Av_2 & \dots & Av_n \\ | & | & & | \end{bmatrix}$$

Matrix Diagonalization

$$AP = \begin{bmatrix} | & | & & | \\ Av_1 & Av_2 & \dots & Av_n \\ | & | & & | \end{bmatrix}$$

$$Av_i = \lambda_i v_i \quad \Rightarrow$$

$$AP = \begin{bmatrix} | & | & & | \\ \lambda_1 v_1 & \lambda_2 v_2 & \dots & \lambda_n v_n \\ | & | & & | \end{bmatrix}$$

Matrix Diagonalization

$$AP = \begin{bmatrix} | & | & & | \\ \lambda_1 v_1 & \lambda_2 v_2 & \dots & \lambda_n v_n \\ | & | & & | \end{bmatrix} =$$

$$\underbrace{\begin{bmatrix} | & | & & | \\ v_1 & v_2 & \dots & v_n \\ | & | & & | \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}}_{\Lambda}$$

Matrix Diagonalization

$$AP = P\Lambda$$

$$P^{-1}AP = P^{-1}P\Lambda$$

$$\Lambda = P^{-1}AP$$

The diagonalization matrix P is formed by the eigenvectors of A.

Symmetric Matrix Diagonalization

If $A = A^T$, then the columns of P are orthogonal and

$$P^{-1} = P^T$$

$$\Lambda = P^T A P$$

The matrix P represents a ROTATION

Example:

$$A = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix}$$

$$\lambda_1 = 1, \lambda_2 = 2 \quad v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad v_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \qquad P^{-1} = \begin{bmatrix} -3 & 2 \\ 2 & -1 \end{bmatrix}$$

$$P^{-1}A = \begin{bmatrix} -3 & 2 \\ 4 & -2 \end{bmatrix} \qquad P^{-1}AP = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Example:

$$A = \begin{bmatrix} 12 & 4 \\ 4 & 12 \end{bmatrix}$$

Symmetric!

$$\lambda_1 = 16, \lambda_2 = 8 \qquad v_1 = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \; v_2 = \begin{bmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$

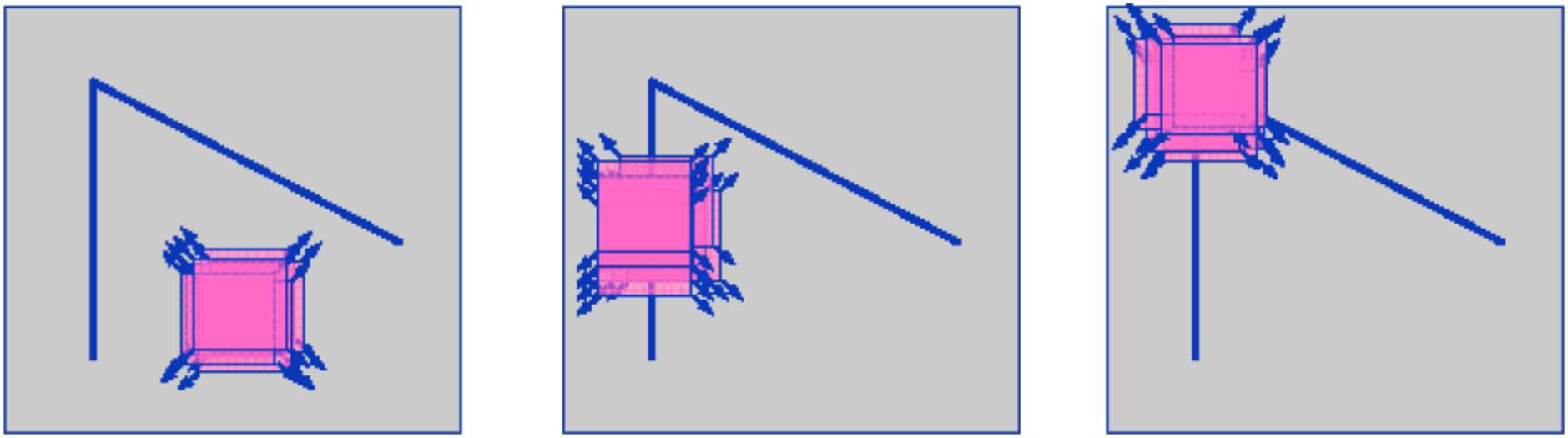
$$P = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \qquad P^{-1} = P^T = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

45° rotation!

$$P^{-1}AP = \begin{bmatrix} 16 & 0 \\ 0 & 8 \end{bmatrix}$$

Harris Corner Detector

The Harris corner detector gives a mathematical approach for determining the amount of changes when we move in all directions a small window.



Harris Corner Detector

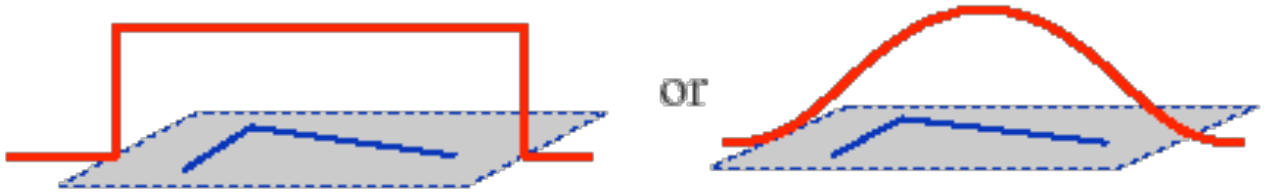
Change of intensity for the shift [u,v]:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

Window Function

Change In Intensity

Window w(x,y):



1 in window, 0 outside

Gaussian

First Order Approximation of f(x,y):

$$f(x+u, y+v) = f(x,y) + uf_x(x,y) + vf_y(x,y) + \ldots$$

Harris Corner Detector

$$\sum_{x,y}[I(x+u,y+v)-I(x,y)]^2 \approx$$

$$\sum_{x,y}[I(x,y)+uI_x(x,y)+vI_y(x,y)-I(x,y)]^2 =$$

$$\sum_{x,y} u^2 I_x^2 + 2uvI_xI_y + v^2 Iy^2$$

Harris Corner Detector

$$\sum_{x,y} u^2 I_x^2 + 2uvI_xI_y + v^2 Iy^2$$

Rewrite as a Matrix:

$$\sum_{x,y} u^2 I_x^2 + 2uvI_xI_y + v^2 Iy^2 =$$

$$\sum_{x,y} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} =$$

$$\begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

Harris Detector

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

M is computed from the gradient components

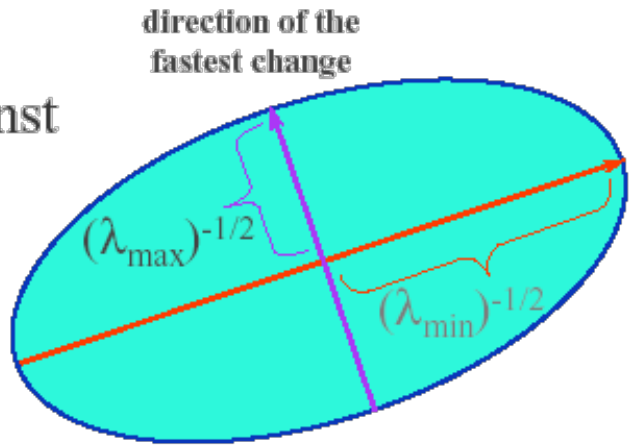
Harris Detector

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Intensity change in shifting window eigenvalue analysis:

λ_1, λ_2 eigenvalues of M

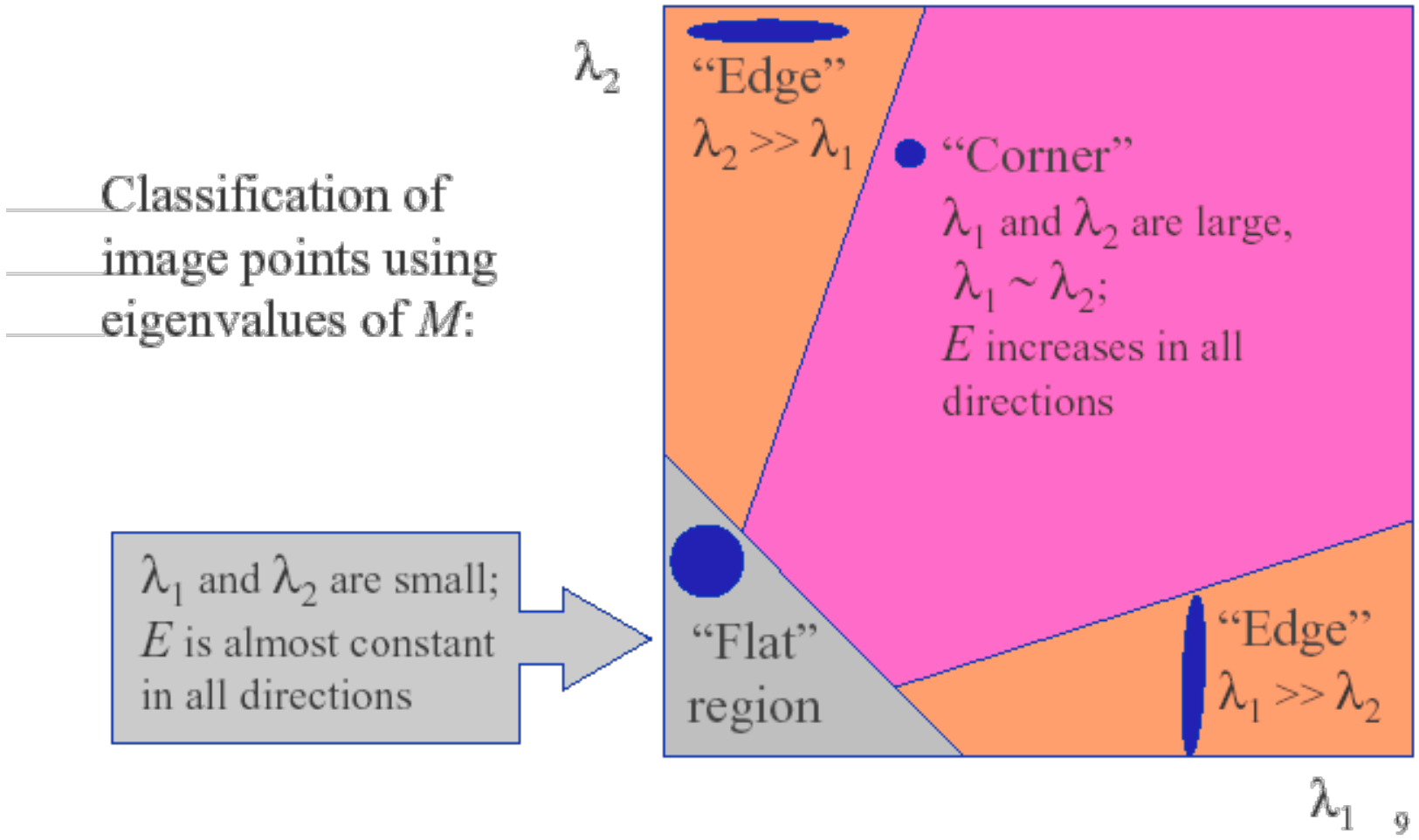
Ellipse $E(u,v) = \text{const}$



8

Classification via Eigenvalues

Classification of image points using eigenvalues of M:



Corner Response Measure

Measure of corner response:

$$R = \det M - k\left(\text{trace}\, M\right)^2$$

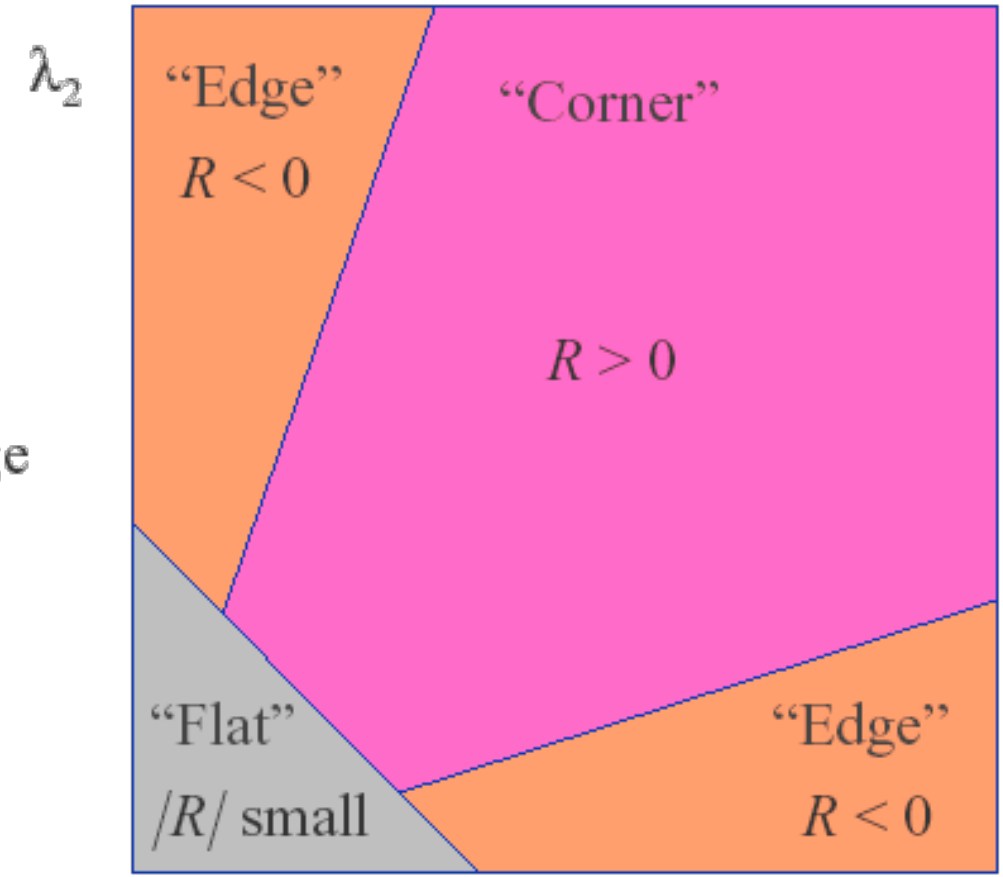
$$\det M = \lambda_1 \lambda_2$$
$$\text{trace}\, M = \lambda_1 + \lambda_2$$

(k is an empirically determined constant; $k = 0.04$ - 0.06)

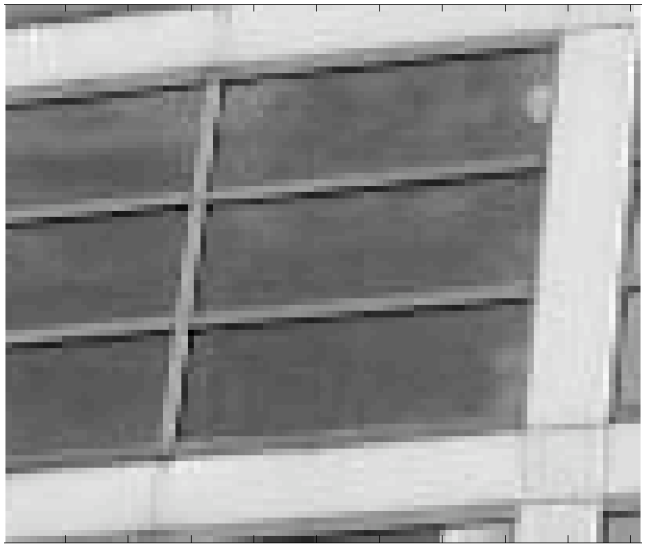
Corner Response Measure

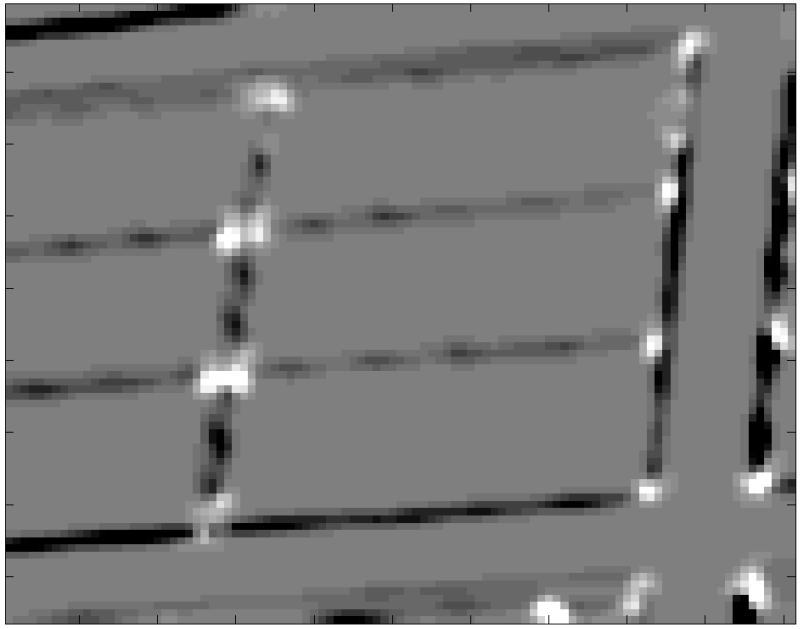
$R = \det M - k \, \text{trace}^2(M)$

- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



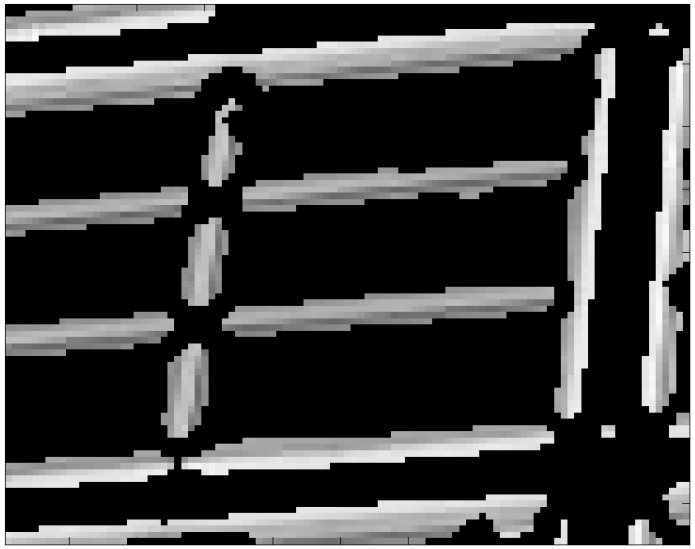
Corner Response Example





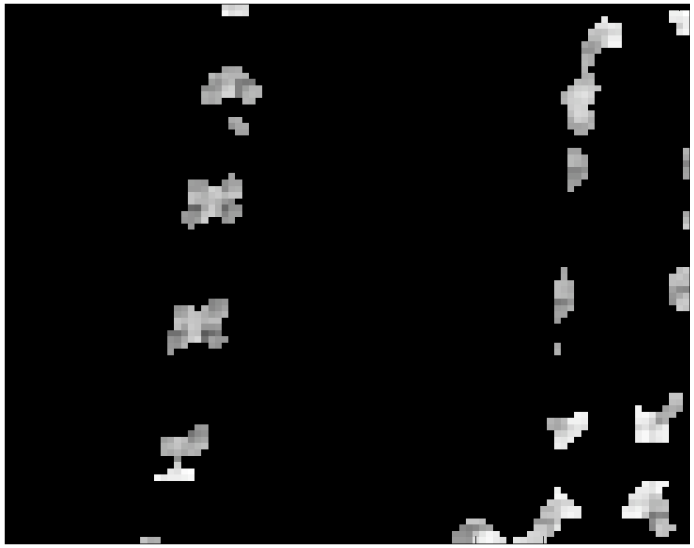
R score; Gradient computed with Sobel mask
Window Gaussian, sigma=1

Corner Response Example: Edges



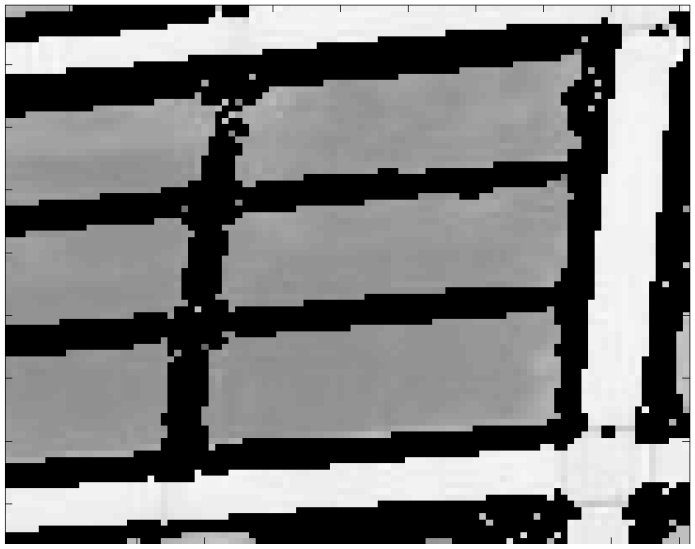
R < - 10000

Corner Response Example: Corners



R > 10000

Corner Response Example:



-10000 < R < 10000

Neither edges nor corners

Harris Corner Detection Algorithm

Compute the Image Gradient

$I_x = G_{x,s} * I$ and $I_y = G_{y,s} * I$

Compute products of derivatives at each pixel

$I^2_x = I_x \cdot I_x$ and $I^2_y = I_y \cdot I_y$ and $I_{xy} = I_x \cdot I_y$

Compute the sums of the products at each pixel using a window averaging:

$S^2_x = G_{s'} * I^2_x$ $S^2_y = G_{s'} * I^2_y$ $S_{xy} = G_{s'} * I_{xy}$

Define the Matrix at each pixel $M = [S^2_x \; S_{xy} \; ; \; S_{xy} \; S^2_y]$

Compute the response $R = \det M - k \, \text{trace}(M)^2$

Threshold R

Compute Nonmax suppression